University of Alberta

A Quality Framework for Small Software Organizations

by

Sundari Voruganti

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 1997

National Library of Canada
Acquisitions and Bibliographic Services
395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale du Canada
Acquisitions et services bibliographiques
395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-22687-5

Canada

University of Alberta

Library Release Form

Name of Author: Sundari Voruganti

Title of Thesis: A Quality Framework for Small Software Organizations

Degree: Master of Science

Year this Degree Granted: 1997

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Sundari Voruganti

Sundari Voruganti
2123, 111A Street,
Edmonton, Alberta,
Canada T6J 4W8

Date: June 11, 1997

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **A Quality Framework for Small Software Organizations** submitted by Sundari Voruganti in partial fulfillment of the requirements for the degree of **Master of Science**.

. . . . . .
Dr. Paul G. Sorenson (Supervisor)

. . . . . .
Dr. H. James Hoover (Co-supervisor)

. . . . . .
Dr. Michael Barrett (External)

. . . . . .
Dr. Ling Liu (Examiner)

Date: 10 June 97.

To Swami,
my Parents, Murthy and Kiran,
Attayya, Pavan and Ryan
and my love Kaladhar

Abstract

Today software plays a critical role in various aspects of human life, from rockets to health care. Not surprisingly, a number of organizations have sprung up in order to meet this need for software. Among these are software organizations that comprise of less than ten people which we term *small organizations*. Although these small organizations have few resources in terms of time, money and people, they often develop leading edge software for new application domains or do important subcontracting work for larger companies. In order to gain acceptability in the marketplace for their products or to obtain contracts from the larger companies, it is necessary for them to produce high quality software in the shortest possible time. A number of standards have been proposed in order to assess and improve an organization's software and process quality. But these standards are geared towards large organizations and are hence too complex for small organizations. The main goal of this thesis is to address this deficiency.

There are three main contributions of this thesis. First, an assessment of a particular small software company, NewCo, is performed to identify the quality issues of small software organizations. Second, a Quality Framework that is affordable, effective, scalable, usable and aids in *out-sourcing* of quality is proposed. The quality framework provides support for performing quality activities in a small software organization. Third, the quality framework is prototyped using a semi-automated tool developed as part of this thesis work in order to help NewCo perform quality activities.

Acknowledgements

First of all, I would like to express my sincere gratitude to my husband Kaladhar for his love, support and patience. He was always willing to listen and read. As my toughest critic, he helped make my work better. My love and gratitude to my parents, Vishwanadham, Subhadra and Syamala for their everlasting love and support. My brothers, Murthy, Kiran and Pavan were always there for me. Thanks to my golden retriever Ryan who helped me see the silver lining behind each cloud.

I am very grateful to my supervisors, Dr. Paul Sorenson and Dr. Jim Hoover for their invaluable guidance and vision during my research. They were always willing to listen to my ideas and gave me the freedom to explore them. Without their help and guidance, this thesis could not have been accomplished. I would like to thank the members of my examining committee Dr. Ling Liu and Dr. Michael Barrett for their helpful suggestions. Thanks to Dr. U.M Maydell for chairing my defense.

I would like to thank Tony Olekshy for taking time off his busy schedule to attend meetings and give feed back.

Special thanks to Narendra Raavi, Gafry Froehlich, Dr. Piotr Findeisen for all the help and numerous discussions. Special thanks to Amr Kamel for all the discussions that led to the spiral model for quality.

Sincere thanks to Adriana and Asha for their friendship and for keeping me sane.

Finally, I would like to acknowledge the financial support offered by Josh Kolenc of Teledyne Fluid Systems-Farris Engineering and the Department of Computing Science, University of Alberta.

Contents

1 Introduction **1**

1.1 Motivation 3

1.2 Outline of the Thesis 4

2 Background **5**

2.1 Small Software Organizations 5

2.2 Characteristics of a Small Software Organization 6

2.2.1 Strengths of a Small Software Organization 6

2.2.2 Weaknesses of a Small Software Organization 7

2.2.3 Differences between a Large Software Organization and a Small Software Organization 7

2.3 Quality System for Small Software Organizations 8

2.4 Process Models for small software organizations 10

2.4.1 Waterfall model 11

2.4.2 Prototyping 11

2.4.3 Spiral Model 14

2.5 Conclusions and Recommendations 14

3 Case Study **16**

3.1 Background - Quality Standards 16

3.1.1 Capability Maturity Model (CMM) 16

3.1.2 ISO 9001 18

3.1.3 Bootstrap 19

3.1.4 Software Process Improvement and Capability dEtermination (SPICE) 20

3.2 Choice of Assessment Approach 22
3.3 NewCo 23
3.3.1 Characteristics of NewCo 24
3.4 SPICE Assessment of NewCo 24
3.4.1 Assessment Plan 25
3.4.2 Assessment Results 27
3.4.3 Initial Improvement Opportunities 28
3.5 Summary 31

4 The Quality Process and Quality Framework **32**
4.1 Quality Process 32
4.1.1 Quality Spiral 0: Base-lining the process 32
4.1.2 Quality Spiral 1-Onward: Process Evolution Cycle 34
4.1.3 NewCo Q uality Spiral 35
4.2 Quality Framework 37
4.3 Properties of the Quality Framework 37
4.4 Defining a Quality Framework 40
4.4.1 Automated Support 42
4.5 Summary 44

5 Components of the Quality System **45**
5.1 Reviews 45
5.1.1 Walk-throughs 47
5.1.2 Inspections 47
5.1.3 Technical Reviews 50
5.2 Metrics 52
5.2.1 Collection of measurements 53
5.3 Interpretation of quality information 55
5.4 Summary 57

6 Design and Implementation of the Quality System **58**
6.1 Requirements of the Quality System 58
6.2 Architecture 59

6.2.1 Database Design 60
6.2.2 Reports 62
6.3 Implementation of the quality system 68
6.4 Conclusions 69

7 Contributions and Future Research **72**
7.1 Contributions 72
7.2 Future Research 74

A User Manual **81**
A.1 Login Form 81
A.2 Options Form 81
A.3 Work-Product Information 83
A.4 Choose the Review 83
A.5 Review Process 87
A.5.1 Review Form 87
A.5.2 Individual Problem Form 87
A.6 Inspections 90
A.6.1 Project Identification Form 90
A.6.2 Inspection Performance Form 90
A.6.3 Individual Problem Report Form 90
A.7 Reports 93
A.8 Change/Query Form 93

B Technology Review **98**
B.1 Requirements for the quality system 98
B.2 Tools under review 99
B.2.1 Lotus Notes 99
B.2.2 Intranet 99

List of Figures

2.1 Experimental Prototyping 12
2.2 Exploratory Prototyping 13
2.3 Evolutionary Prototyping 15

4.1 The Spiral Model for Process Improvement 33
4.2 The Spiral Model for Process Improvement for NewCo 36
4.3 A model for interactions between Development and Quality processes 41
4.4 Work-product interaction between development and quality processes 43

5.1 Sample Company Profile 56

6.1 Entity-Relationship Diagram 61
6.2 Overview 69
6.3 Reviews and Inspections 70

A.1 Login Form 82
A.2 Options Form 84
A.3 Work-Product Form 85
A.4 Choose Form 86
A.5 Review Information Form 88
A.6 Individual Problem Form 89
A.7 Project Identification Form 91
A.8 Inspection Performance Form 92
A.9 Defect Form 94
A.10 Reports 95
A.11 Process Improvement Reports 96
A.12 Change/Query Form 97

List of Tables

6.1 Problem Table 60
6.2 Reason Table 62
6.3 Work-product Table 62
6.4 Inspection Table 63
6.5 Review Table 63
6.6 Measurements Table 63

A.1 Login Form 82
A.2 Work-product Form 83
A.3 Problem Form 90

Chapter 1

Introduction

As computing power continues to increase, software solutions are being developed for newer application domains. Presently, software plays a role in various aspects of human life, from rockets to health care. Not surprisingly, a number of organizations have sprung up in order to meet this need for software. Among these are *small organizations* that comprise less than ten people. Typically, these organizations have few resources in terms of time, money and people. The small organizations most often develop leading edge software for new application domains or do subcontracting work for larger companies. In order to gain acceptability in the marketplace for their products or to obtain contracts from the larger companies, it is necessary for them to produce high quality software in the shortest possible time.

A decade ago, software quality was determined mostly by its functionality. Competition and the need for highly reliable computer systems have forced software suppliers to create quality software that does more than simply meet its functional requirements. Quality, in the broader sense, is defined in the International Standard Quality Vocabulary [ISO86] as:

The totality of features and characteristics of a product or service that bear on its ability to meet stated or implied needs.

This notion of quality is heavily based on customer satisfaction, assuming that customers will be satisfied if their stated and implied needs are met. In order to satisfy customers consistently, quality must be built into the software process through

all the phases of the product life cycle, from requirements capture to development and support [SC94].

Quality may be measured by customer satisfaction, but more tangible methods of measuring it have been identified and used. Assessment of the process used to develop a product provides an indication of the quality of the product and is one of the methods used to measure quality. Several assessment methods have been proposed and several are used to assess and improve the quality of the software process. The Capability Maturity Model (CMM)[PWG93], Software Process Improvement and Capability dEtermination (SPICE) [Dor93] and ISO 9001 [cou94] are a few of the popular international standards that have been developed to assess software process quality. But these standards are geared towards large organizations and hence are too all encompassing for small organizations. This thesis investigates this gap and attempts to fill it with a quality framework for small software organizations that provides the necessary support for performing quality activities.

In this thesis, a model is defined and tools created to support process improvement for a small-software organization. The specific objectives of our study are to:

1. define the characteristics of a small software company;
2. assess a small company to evaluate its current process quality based on a suitable software standard;
3. identify the areas that need improvement;
4. develop a quality framework that supports the quality activities in an organization;
5. prototype a quality system that is an instantiation of the quality framework and provides functionality for:
   (a) collecting measurements;
   (b) performing quality activities such as reviews/inspections;
   (c) performing individual problem tracking and traceability in support of root cause analysis;

(d) providing reports for analysis.

Section 1.1 discusses the motivation for this thesis in greater detail. Section 1.2 outlines the remaining chapters of the thesis.

1.1 Motivation

The motivation for this research is the fact that small software organizations have few resources in terms of time, people and money and therefore cannot perform many of the recommended quality activities. The small software organization may need to comply with given standards if they want to sub-contract with larger organizations. Small organizations recognise the fact that the quality of a product depends greatly on the kind of the process used to produce that product. But they do not have the resources to perform quality activities recommended by existing quality standards. Progress from an assessment to the implementation of a quality process requires guidance (as in the CMM). Tool support, as provided for example in EssentialSET[1] is also desirable.

The objectives of this research are to :

- propose a quality model that is effective, affordable, scalable and usable;
- examine the feasibility of a small software organization *out-sourcing* its quality assessment and improvement activities. That is, an outside agency would perform these activities for the organization so that they can concentrate on product development.

A number of questions are considered in this investigation. Three of the most important are:

- What are the quality activities that can be enacted by a small software development organization without significant drain on essential resources?
- What is the tool support that can be provided in order to perform the quality activities?

[1]EssentialSET is a trademark of Software Productivity Centre.

- Can the interfaces between the quality and development activities be clearly defined so that small organizations can *out-source* their quality activities?

These issues are a major focus of the thesis and are discussed in the following chapters.

1.2 Outline of the Thesis

This thesis is organized as follows. Chapter 2 presents an overview and definition of small software organizations and their characteristics. The second half of the chapter focuses on the quality system and process models pertaining to small orgahizations. This chapter concludes with recommendations for small organizations about the development models to use. In chapter 3, an overview of the current standards is presented. This chapter provides the SPICE assessment of NewCo as a case study and identifies improvement opportunities. Chapter 4 provides our quality process and our proposal for a quality framework that is effective, affordable, scalable and usable. Chapter 5 outlines the components of the quality system based on the SPICE assessment. Design and implementation aspects of the quality process are described in Chapter 6. Finally, the thesis concludes in Chapter 7 with a summary of the major contributions of this research, followed by suggestions for future research in this area.

Chapter 2

Background

This chapter presents the main characteristics of small software organizations and process models that apply to them. Section 2.1 and 2.2 gives a general classification of small companies. Section 2.3 discusses the necessary properties of a quality system for small software organizations. Section 2.4 discusses the generic process life-cycle models.

2.1 Small Software Organizations

Small companies have been defined in a number of ways. The Canadian government defines a small business as[SS88]:

> *A small business is a firm whose gross annual revenue is less than $2.0 million and has less than fifty employees in a service sector or less than hundred employees in a manufacturing sector.*

This thesis focuses on small organizations in the business of developing software products. The type of products developed by a small software organization may be divided into risky and non-risky ventures[Coo94]. Risky ventures are those products that are new to the world and hence complex. Examples are the initial version of MS-DOS or Java. Application domains where the company does not have previous experience or expertise also fall into this category.

Low risk ventures are somewhat less ill-defined, such as improving or revising existing product lines, or sub-contracting to a large software organization to develop

software subject to certain software standards and/or well-understood algorithms.

2.2 Characteristics of a Small Software Organization

In this thesis, a small software organization is defined based on the number of people in the organization because this measure offers the following advantages [Sir82]:

- Inflation Proof: unaffected by changes in the purchasing power of the dollar;
- Transparent: easy to see and measure;
- Comparable: allows comparisons in size between companies in the same industry;
- Available: the number of employees in a company is easily available whereas information such as yearly revenues may not be as easily available.

The Corporate Directory [Inc96] listing of all computer organizations for the city of Edmonton in the year 1996 showed that approximately 78% of nearly 400 organizations had 9 employees or less. Even if 10% of these organizations perform non-software development functions, there is still a large percentage of small organizations producing software.

Specifically, we define a small software organization as:

A small software organization is one that is a primary producer of relatively new software and which has between four and ten employees.

2.2.1 Strengths of a Small Software Organization

A small software organization has the following strengths:

- employees perform cross-functional tasks;
- they have a flat organizational structure (that is, there is little or no management hierarchy);

- communications are informal but effective. Each member usually knows what the other members of the team are working on;
- more hours are likely to be *donated*. That is, team members work longer hours without getting paid overtime but may be remunerated with shares in the company;
- very efficient in the use of resources because each employee is expected to be actively involved in the company activities at all times;
- responds much more quickly and with less cost to changes in the market needs.

2.2.2 Weaknesses of a Small Software Organization

Small software organizations also have limitations:

- fewer resources in term of time, money and people;
- generally, not many activities take place in parallel;
- communications and management are informal and therefore important decisions may not be recorded;
- quality activities generally take a back seat because of the drain on resources to verify quality;
- the loss of team members can incapacitate the project; and
- new opportunities are difficult to undertake because they require a large number of additional resources.

2.2.3 Differences between a Large Software Organization and a Small Software Organization

- Large software organizations have a lot of resources in terms of people, money and time.
- A lot of activities take place in parallel in large software organizations. For example, quality assurance is typically carried out by a group separate from the

development group. By necessity, in small software organizations, the quality assurance and development activities are performed by the same people.

- Large software organizations have a well-defined organizational hierarchy.
- Teams work more independently in large software organizations and typically do not perform cross-functional tasks. Therefore, one set of team members may not be aware of what other team members are doing.
- The communication lines are more formal in large software organizations. There are defined protocols for many of the communications.
- There is an opportunity and usually an expectation for the development and quality activities to be well defined and well documented in large software organizations.

2.3 Quality System for Small Software Organizations

A software process is a sequence of steps required to develop or maintain software [Hum95]. Specifically, a software process provides the technical and management framework for applying methods, tools and people to the software tasks. A process definition is the detailed definition of the software process. Defining and developing the process is a time-consuming task. In principle, by using processes defined by a previously successful project, time and resources can be saved. As a process is enacted (used), it may need changes and it is only a well defined process that can be improved. For example, some parts of the process may be badly defined, or some steps may not be followed. Changes to the process can then be made to address the problems. Thus the process improves with use and experience. The software process quality is defined by how well the process definition fits the actual process. The software process definition should identify what quality assurance activities have to performed and how. The quality activities may be reviews, inspections, testing activities or collection of measurements. For example, developing code from design is a development activity. Performing reviews on this code is a quality activity. Mea-

surements have to be collected in order to measure the effectiveness of the quality activities. Based on these measurements, changes may be made to the quality activities. In large software organizations, a separate group performs the task of developing quality and development processes, and improving them.

A small software organization has few resources in terms of time, people and money and cannot afford to have a separate quality group. But current software standards assume the availability of a number of resources to the organization that wants to perform the recommended quality activities. Since small software organizations have limited resources, a quality system for such organizations must, by necessity, be effective, affordable, scalable and usable.

Effective

Webster's dictionary [Web88] defines effective as:

causing or capable of causing a desired or decisive result

Assuming proper use of the quality system, its effectiveness can be measured during a formal assessment of the organization. Since the organization's processes will mature as the quality activities are performed, comparison of assessment results before and after the use of the quality system will provide a measure of effectiveness of the system.

Affordable

The quality activities that are suggested to a small software organization must be within their resource limits. The quality activities should help a small organization do their development work better. For example, if for every 100 hours of development work, an organization performs 10 hours of quality activities, the result should be equivalent to at least 120 hours of development work.

In order to enable small software organizations to perform quality activities, they should be supported by tools that help in automating some of the tasks. Any quality system that is proposed should not have too much overhead in terms of learning curves or startup time.

Scalable

Scalability may be defined as the degree to which architectural, data or procedural design can be expanded [Pre92].

As the organizations grows by hiring more people, getting more projects, increasing its process maturity, the quality requirements change. For example, as the organization's processes mature, different types of measurements will be collected. The quality system should be easily extendable and flexible to meet the changing needs of the organization.

Usable

The ISO (International Standards Organization) definition of usability is the *effectiveness, efficiency, and satisfaction with which a specified set of users can achieve a specified set of tasks in particular environments.* This ISO definition is operational in nature, requiring task and user definition, and requiring the means for measuring effectiveness, efficiency, and satisfaction.

A common sense definition of usable is a software product that is easy to learn, efficient to use, recovers quickly from errors and is easy to remember. The tool support that is provided for the quality system must be easy to use and must have a low learning curve. The tool should not be perceived by the developers as being intrusive.

2.4 Process Models for small software organizations

A defined process model is an essential component of well-managed software development. This principle is fundamental to the assumptions behind ISO 9001 and SEI's Capability Maturity Model. A software organization's process model depends on the type of applications/product that are being developed. If the software organization develops products in areas where the specifications are stable, for example, text editors or compilers, the organization may use a waterfall model [Roy70].

But if the organization is developing products in areas where the requirements are not fully understood or if the team does not have expertise in the application domain, a risk assessment based model such as the spiral model [Boe88b] should be

used along with prototyping to further reduce risk. A brief overview of the process models is give below.

2.4.1 Waterfall model

The waterfall model [Roy70] views the software process as a series of activities performed sequentially: requirements specification, design, implementation, testing. After each phase has been defined, development proceeds to the next stage.

The waterfall model is a serial model, that is, development moves from one stage to another only after the previous stage is fully completed. In most cases, the requirements are not always clear at the start of a project and change frequently. Therefore, even if project development continues under the assumption that all requirements are stable, changes in requirements force iteration through all the phases again. Therefore, a waterfall model is optimum only in application domains where the requirements are stable or well defined at the start of the project.

2.4.2 Prototyping

A software prototype is a partial model of a system used to investigate the system requirements and specification and to test the feasibility and reliability of the design [MW91]. It helps in the discovery of missing or faulty requirements and to avoid the unnecessary use of resources in developing the wrong product [Wal94]. Prototyping is also an effective technique of risk management [Wal94] because it prevents the commitment of major resources without evaluation. Since small software organizations do not want to waste resources developing the wrong product, such risk management techniques are recommended [Boo95]. There are different approaches to prototyping [MW91]:

Experimental Prototypes

Experimental prototyping [Wal94] is used to clarify the design specifications that form the basis for implementation. These types of prototypes are used to support system design. These prototypes involve mostly the developers who wish to evaluate the architecture of the system. Such prototypes are generally thrown away. The advantages of experimental prototyping are[Wal94]:

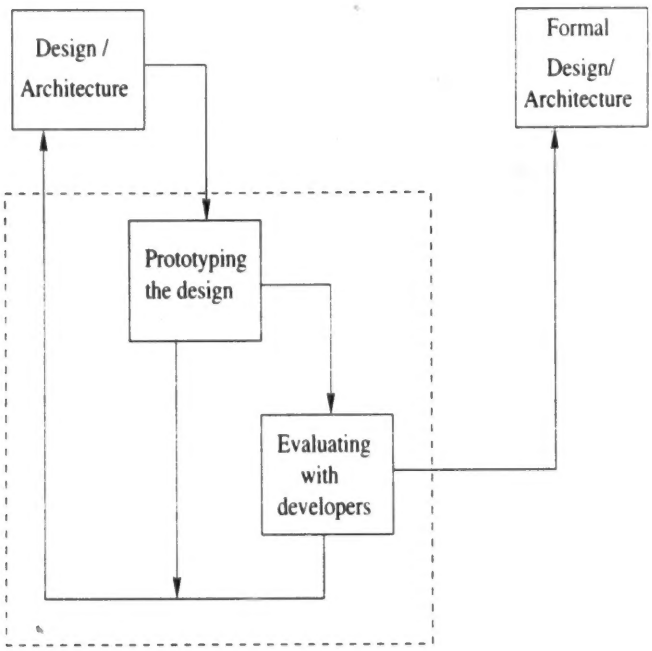

Figure 2.1: Experimental Prototyping

- early testing of system architecture and design;
- verification of intermodule interaction;
- consistent and complete specification of system components; and
- an extension to static design reviews.

A major disadvantage of experimental prototype development is the additional cost and effort involved which can delay the project.

Exploratory Prototypes

Exploratory prototyping (see Figure 2.2) is used to clarify parts of the system and to examine alternative solutions. Exploratory prototyping is informal and unstructured and the prototypes are generally thrown away. The advantages are[Wal94]:

- Specification problems are dealt with early in the project at a lower cost.

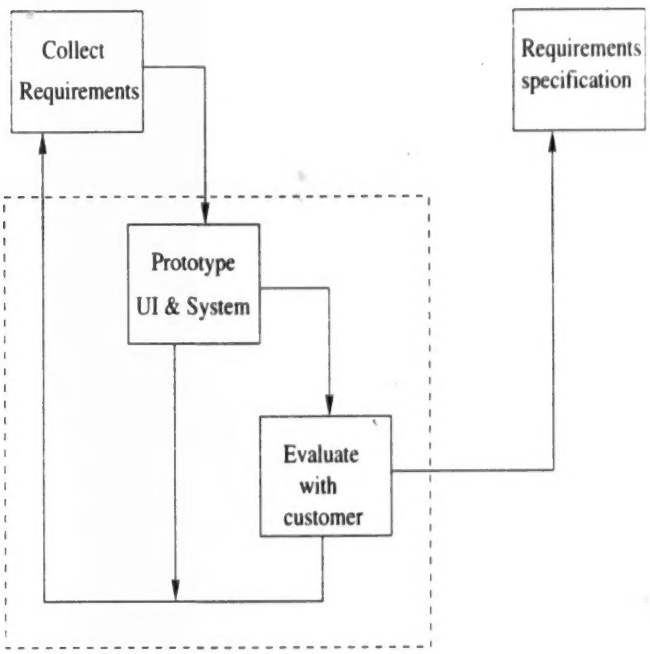

Figure 2.2: Exploratory Prototyping

- Besides the user interface, parts of the functionality of the system are also modelled and tested.
- The language and communication problems between the developer and user are eliminated.
- The user's goals and needs are validated with a working system.

The disadvantages are:

- There is a danger of the user assuming that the prototype is the real system.
- If the prototype becomes part of the final product, the system might become brittle at a later stage. This is because no quality assurance has been performed for those parts of the system.
- A small software organization has very few resources and a throwaway prototype wastes these resources.

Evolutionary Prototyping

Evolutionary prototyping depicted in Figure 2.3 is used to refine a solution gradually. The prototype is refined as the nature of the problem is revealed over time. Therefore, a fully functional system is iteratively built and tested as new requirements are uncovered. The advantages are[Wal94]:

- The prototype is the manageable part of the system specification. Therefore, the problem of incomplete or faulty requirements can be solved using this technique.
- Maintaining code and documentation is easier because maintainence requirements are in sight from the start.
- Quick validation of requirements is possible on a working system.

This technique has the disadvantage that it is very sensitive to the quality of the architecture and therefore must be designed properly. The process of building the system must follow a life-cycle model incorporating quality activities, otherwise changes will just be patched into the system which makes it brittle in later stages.

2.4.3 Spiral Model

The spiral model[Boe88b] is a risk driven model in which tasks are evaluated relative to their risk. Prototyping may be used as a risk reduction mechanism. This model may encompass other models as parts of its spiral.

Since software packages are products, product development process models can be used to develop software. Cooper's model [Coo90] is an iterative model where the technical and marketing activities take place in parallel. There are frequent evaluation points, called *gates*, between stages where "Go/No Go" decisions are made. The principal stages in the model include: idea generation, preliminary assessment, secondary assessment, development, testing, and launch activities.

2.5 Conclusions and Recommendations

In summary, a small software organization has few resources in terms time, money or people. As a result, any quality system that is introduced must be effective, affordable, scalable and usable.

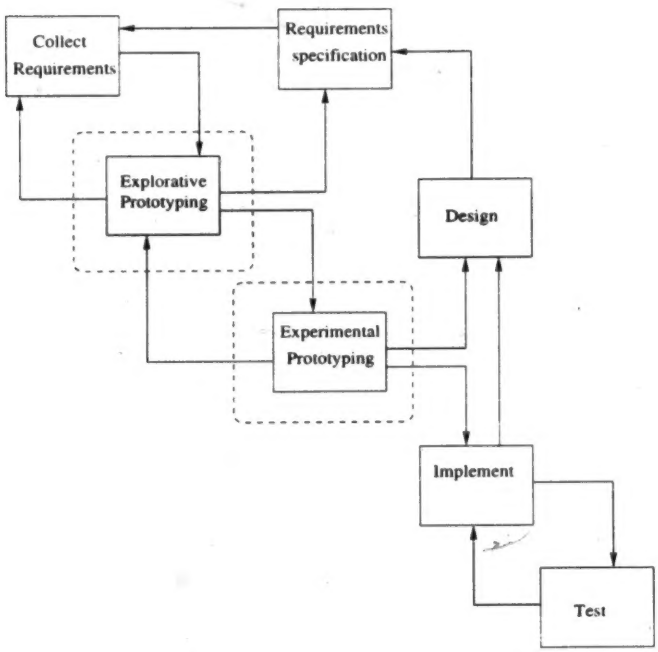

Figure 2.3: Evolutionary Prototyping

We argue that a small software organization developing new products where the requirements are not fully understood should use a spiral model with evolutionary prototyping. This is because the spiral model helps in assessing and mitigating risk. The evolutionary prototype method helps in incrementally adding new requirements and improving design.

Evolutionary prototyping has the advantage that all requirements need not be known at the start of the project. The subset of the requirements that are understood are built into the prototype in the first release. The remainder of the requirements are subsequently built into the system using iterative refinements of the prototype. Small software organizations do not have enough resources to build and then throw away prototypes. Therefore, evolutionary prototyping is recommended.

Chapter 3

Case Study

This chapter provides an overview of quality standards and the SPICE assessment of NewCo. Section 3.1 provides an overview of several current standards including the Capability Maturity Model, ISO 9001, Bootstrap and SPICE. Section 3.2 outlines why SPICE was chosen. Section 2.3 describes the characteristics of NewCo, the company undertaking the SEAF[1] project development. Section 3.4 and 3.5 outline the formal SPICE assessment performed on NewCo.

3.1 Background - Quality Standards

Software organizations that want to compete in today's market need to have in place a mature software development process or mechanisms to progressively improve their software development process. That is, not only do they need to have a process, but they need a way to improve that process. To this end, a number of assessment methods have been proposed and used in order to assess process quality. The purpose of these assessments vary from wanting to establish that an organization has adopted some of the best practices (eg. ISO 9001) to wanting to identify areas of process improvement (eg. CMM). Some of the popular methods are outlined in this section.

3.1.1 Capability Maturity Model (CMM)

The Capability Maturity Model for software [PWG93], developed by the Software Engineering Institute(SEI), is a framework that describes an evolutionary improvement path from an ad-hoc development process to a mature disciplined development

[1]Size Engineering Application Framework

process. The CMM recommends practices for planning, engineering and managing software development and maintenance processes which improve the organization's ability to meet goals for cost, schedule, functionality and product quality. The CMM is organised into five maturity levels. The five levels of maturity are:

1. **Initial:**

   The software process is characterized as ad-hoc and occasionally chaotic. Few processes are defined and success depends on individual effort and heroics.

2. **Repeatable:**

   Basic project management processes are established to track cost, schedule and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

3. **Defined:**

   The software process for both management and engineering activities is documented, standardized and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.

4. **Managed:**

   Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled.

5. **Optimizing:**

   Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.

Each maturity level is decomposed into several key process areas that indicate where an organization should focus in order to improve its overall process. Each key process area is described in terms of key practices that contribute to satisfying the goals of the area. The key practices describe the infrastructure and activities that need to be performed for effective implementation of the area. A Software Engineering

Process Group (SEPG) is recommended that oversees the process activities in an organization.

The CMM has a number of disadvantages[Sil92]: it institutionalizes quality assurance and process groups, does not provide any software support and provides no quantitative process-performance metrics.

The CMM does not address issues that pertain to expertise in a particular application domain; nor does it advocate specific tools, methods or software technologies. It does not address issues related to human resources, concurrent engineering, teamwork, change management or systems engineering [SK95].

3.1.2 ISO 9001

ISO 9001, developed by the International Organization for Standardization (ISO), is the management system standard for quality in manufacturing. A key goal is the establishment of uniform methods of quality assurance across international borders. The standardized quality assurance programs are intended to result in less costly, more reliable products for customers. ISO 9000-3 is the guide for interpreting the ISO 9001 guidelines as they apply to *the development, supply and maintenance of software*. The only meaningful certification for a software producer is ISO 9001/9000-3.

ISO 9001 assumes that products are manufactured in a formal contractual environment where the specifications are correct and do not change[Coa94]. But this is generally not true in the development of software products. Software products are complex and inherently hard to scope, develop, verify and maintain. Therefore, there is a need for a quality approach focused on continuous improvement. ISO 9001 does not support continuous process improvement[Coa94] and partially addresses areas that are important to the development of a good quality software product: human resources, management commitment, marketing, processes, technology and capital. ISO 9001/9000-3 certification is heavily dependent on the experience of the auditor and is awarded if the auditor is satisfied that the processes are in place. The certification does not provide any detailed information as to which processes are mature and which of them need to be improved.

3.1.3 Bootstrap

Bootstrap[HMK+94] was a project performed as a part of the European Strategic Program for Research in Information Technology (ESPRIT). Its goal was to develop a method for software process assessment, quantitative measurement, and improvement. A basis for Bootstrap was the adaption of the SEI's Capability Maturity Model to the needs of European non-defense industries such as banking, insurance and administration. Bootstrap is a method that could be applied to a number of *software producing units*, including small to medium sized software companies or departments within a large company.

The Bootstrap method has the following major elements:

- a hierarchy of process-quality attributes from ISO 9000-3 guidelines for software quality assurance and European Space Agency's PSS05 software engineering standards.
- a refinement of the SEI approach to maturity level calculation for each process attribute.
- an enhanced SEI questionnaire that can be used as a part of the determination of an organization's capability for quality attributes.

The Bootstrap method uses questionnaires based on the ISO 9000-3 guidelines so that organizations can assess their compatibility with ISO 9000-3. Three different kinds of questionnaires are used to extract information about the organization, its methodology and technology[Koc93].

Though Bootstrap is based on the SEI methodology, there are a few basic differences. The CMM is a sequential methodology, that is, scores on the next capability level are considered only if nearly all key questions are answered affirmative on the previous levels. Bootstrap assesses all the processes at all levels. For example, suppose an organization has an efficient methodology and a standardized way of creating documents, but does not have an efficient project management method. With the SEI methodology, the organization would be at a level 1 overall; but with Bootstrap method, it might be rated a level 1 for project management and level 3 for

design[HMK+94]. This level of detail makes it easier to identify and plan improvements.

ISO 9000-3 guidelines do not cover risk management and modern programming practices such as reuse, library management and user interface prototyping. Bootstrap is widely used in Europe.

3.1.4 Software Process Improvement and Capability dEtermination (SPICE)

SPICE [Dor93] is an international collaborative project under the auspices of the International Committee on Software Engineering ISO/IEC JTC1/SC7 [2] through the software process assessment group, Working Group 10 (WG10). The project was established in 1993 and provides a framework for software process assessment. It also embodies a sophisticated model of software process management drawn from the world-wide experience of major Software Process Method suppliers, such as SEI (Software Engineering Institute), and European Bootstrap Consortium, as well as the experiences of Bell Canada and British Telecommunication.

The SPICE document set is a nine part document providing among other things:

- a framework for determining key process strengths and weaknesses;
- a framework for improving the software process and to measure such improvements; and
- a framework for determining the risks of a business considering the development of a new software product or service.

The embedded reference model is based on the principle of examining the practices used to implement the process to determine the capability of that process. The practices are grouped in two categories: *Base Practices*, that are specific to the process being assessed and *Generic Practices* that apply to all processes. Base Practices provide an indication of achievement for the process. On the other hand, Generic Practices indicate how *well managed* the process is.

[2] International Organization for Standards/International Electrotechnical Commission Joint Technical Committee 1 (responsible for Information Technology)/Sub Committee 7 (responsible for Software Engineering).

The model is a two dimensional. In the process dimension, it defines 29 different processes grouped in five categories: Customer-Supplier (5), Engineering (7), Supporting (8), Management (4) and Organizational (5). On the other dimension, the capability dimension, the model defines nine different capability attributes, grouped into six Capability Levels: Incomplete, Performed, Managed, Established, Predictable and Optimizing.

The main difference between ISO 9001 and SPICE is that ISO 9001 is an audit whereas SPICE is an assessment mechanism. The main differences between assessments and audits are[ESI96]:

- An ISO 9000 audit is generally performed by a group external to the organization. Audits have to be performed by independent auditors because the standard provides a set of abstract criteria which have to be interpreted by the auditor contextually. The SPICE reference model has a detailed set of criteria that provide guidance for the assessment. These reduce the number of judgements made by the assessor.

- An assessment takes a thorough look at all the processes in the organization. Depending on the time allocated for the assessment, only a few processess may be examined. Therefore, an assessment focuses on positive evidence. An audit is typically of short duration and as a result focuses on negative evidence. Audits are generally viewed as fault finding exercises and auditors have to get information by careful questioning in a confrontational manner. Most assessments are voluntary and the participants are therefore more cooperative.

- ISO 9001 is a certification, which means that if the auditor finds any non-compliances, certification is not granted; otherwise it is. An organization may achieve a positive result by obstructing and smoke-screening. In an assessment, improvement opportunities are the result, therefore smoke-screening is less likely to achieve anything.

The ISO 9000 series standards specify the minimum requirements for a quality system. SPICE is a process improvement methodology. The ISO 9000 series specify

a set of generalized, abstract requirements whereas SPICE specifies a detailed set of process indicators.

The CMM and SPICE have similar applications, but there are several differences between them:

- CMM is organised into five maturity levels: Initial, Repeatable, Defined, Managed and Optimizing. SPICE is organized in six levels. The extra level, called Incomplete is added.
- The CMM levels apply to a complete organization whereas the SPICE levels apply to a single process.
- In the CMM, the sequence of improvements is built in because an organization is not assessed for higher levels until the lower capability levels are completely satisfied. That is, an organization cannot be assessed for Level 3 if it has not fully satisfied Levels 1 and 2. As a result, it is clear what process areas an organization has to concentrate on in order to move to the next level. But this sequence may not be the best for an organization. The SPICE Model is two dimensional: each process in any category can be at any level of capability.
- The CMM provides a single figure as the capability level of an organization; SPICE provides a process profile. The CMM assumes uniformity of processes in an organization. This is generally not true. In small software organizations, for example, the implementation processes are likely to be at a higher capability level than supporting processes such as reviews, testing training, etc. A CMM assessment does not reflect this disparity. A SPICE assessment provides a process profile of all the assessed processes.

Bootstrap uses questionnaires as an assessment method whereas any method of data collection can be used for a SPICE assessment. The SPICE model is life-cycle independent whereas Bootstrap is tailored to the waterfall life-cycle model.

3.2 Choice of Assessment Approach

The CMM is geared towards large organizations and assumes uniformity of processes across the organization. It does not take into consideration aspects that are key to

a small organization: highly competent staff, use of tools and how to use technology and automation to increase process efficiency[BM91]. An assessment profile is more useful for an organization that wants to improve its process; a capability level does not provide the same information. A study conducted by Johnson and Brodman [BJ94] found that small businesses could not implement CMM because of a shortage of resources. The research found that many of the CMM practices are not applicable to small projects. Another study conducted on a medium sized company [CFL95] showed that companies are well aware of their short comings and are more interested in *how* to put the improvements in place. This area is not very well addressed by the CMM.

The SPICE assessment, on the other hand, evaluates each individual process in an organization, rather than a whole organizational unit. The assessment considers the use of tools and technology. Since we are dealing with a small organization that does not have a defined process, SPICE assessments seemed much more useful for the current process improvement goals of NewCo and was therefore selected.

3.3 NewCo

A Size Engineering Application Framework is currently being developed by a startup, small software organization in co-operation with the Software Engineering Lab at the University of Alberta and an industrial partner. The small software organization, NewCo, is an outgrowth of an engineering and manufacturing firm, Teledyne Fluid Systems - Farris Engineering based in Edmonton. Its mandate is to produce engineering tools for the petrochemical industry. The project was divided into a set of parallel sub-projects [Ole96].

- **Avra**, the *Small Enterprise Quality Process* product, is a System Development Process (SDP) model for applying modern software engineering techniques and technologies to *programming in the small to medium*.

- **Soros** is the suite of engineering tools that will be produced by NewCo. Soros encapsulates the goal of applying our research results to product development.

- **Soros/SM** is the principal test project. This is a product instantiated from the developed framework.
- **Kalos** is the system architecture and application framework technology for Soros.
- **Libras** is the distribution mechanism for tools and techniques produced by NewCo.
- **Mesa** is the group-ware and development environments used by NewCo developers and quality specialists.

3.3.1 Characteristics of NewCo

Apart from the characteristics of small organizations defined earlier, NewCo has the following specific characteristics:

- During pre-beta prototype development few quality activities are performed.
- The development environment of NewCo is distributed. Each employee is at a different physical location and communication is achieved via e-mail, phone and internet technology. Occasional face-to-face meetings also occur.
- NewCo has between four and six employees.]
- A non-functional requirement and quality goal of NewCo is ISO 9001 certification.

Since the company recognizes the importance of a quality process in order to produce a high quality product, a process improvement program was launched as the product release phase of development commenced.

3.4 SPICE Assessment of NewCo

The following section outlines the context of the assessment, its objectives and results. The Assessment of NewCo was performed using the SPICE standard. Three assessors performed the assessment of which one was a second party assessor and the others

were part of the quality team at NewCo. Information was collected by one-to-one interviews and verified using available documentation.

At the time of the assessment, NewCo was not fully staffed and the product was in the pre-beta stage.

3.4.1 Assessment Plan

The objectives of this assessment are:[KV96].

1. Provide detailed information to support the ongoing process improvement initiatives at NewCo; more specifically to:
   - establish a baseline for the current state of practice at NewCo;
   - identify measurable targets for the process improvement program.
2. Provide information to support building an ISO 9000-conformant quality system.

Assessment Scope

In the assessment the following processes are assessed up to capability Level 5:

- *Customer-Supplier Processes:*
  - CUS.1: Acquire Software
  - CUS.2: Manage Customer needs
  - CUS.3: Supply Software
- *Engineering Process Category:*
  - ENG.2: Develop software requirements
  - ENG.3: Develop software design
  - ENG.4: Implement software design
- *Support Process category:*
  - SUP.1: Develop documentation

- SUP.2: Perform configuration management
- SUP.3: Perform Quality Assurance
- SUP.4: Perform work product verification
- SUP.5: Perform work product validation
- SUP.6: Perform joint reviews
- SUP.7: Perform audits
- SUP.8: Perform problem resolution

- *Management Process:*
  - MAN.1: Manage the project
  - MAN.2: Manage quality
  - MAN.3: Manage risks

- *Organization Process:*
  - ORG.1: Engineer the business
  - ORG.2: Define the process
  - ORG.3: Improve the process
  - ORG.4: Provide skilled human resources
  - ORG.5: Provide software engineering infrastructure

Assessment Constraints and Outputs

Only one instance of each process was assessed. The instance for assessing the processes in the Customer-Supplier, Engineering, Management and Supporting categories was chosen from the application area of NewCo and is Soros/SM. The instance for assessing the processes in the Organizational category was chosen from NewCo's management projects.

The following deliverables were available on completion of the assessment:

1. Process profiles for the assessed processes

2. Assessment context statement

3. Improvement opportunities report

4. Assessment final report

3.4.2 Assessment Results

The key findings of the SPICE Assessment were:

- 4.5% of the assessed processes are at Capability Level 3 (Established), 13.6% of the processes are at Capability Level 2 (Managed), 36.4% of the processes are at Capability Level 1 (Performed) and 45.5% of the processes are at Capability Level 0 (Incomplete).

- All applicable Engineering Processes were at Capability Level 1 (Performed). These findings indicate the adequacy of the process to develop a specific product.

- All applicable Customer-Supplier processes are at Capability Level 3 (Established) which was the best overall capability level achieved in the assessment. These findings apply to an in-house customer only.

- All applicable Supporting Processes have the lowest Capability levels - 75% of the processes were at Level 0 and 25% at Level 1. Based on this assessment, later phases of development of the projects are associated with high risk.

- The assessment participants are very enthusiastic about process improvement which indicates a high degree of success for any process improvement program.

The assessment results also indicated specific areas of improvements that the company can concentrate on in the short term. These improvements will be discussed based on organizational needs, expected challenges and high risk areas. The areas identified were documentation, quality assurance, and handling customer requirements.

3.4.3 Initial Improvement Opportunities

1. **Documentation**

   Suggested improvements in this area directly map to the SPICE process SUP.1 (Develop Documentation). Lack of documentation is a major problem the company faces right now. The potential problems are the high learning curve for new recruits as the organization is expecting to expand its programming staff soon and the loss of rationale behind the design and coding decisions.

   Currently, the project does not have an overview of the required document suite. An overview of all the documents that are to be produced in connection with a project is needed. This overview should describe the strategy that steers all other documentation efforts. The suite of documents, as well as the contents of each document, has to be specified in order to properly evolve the suite.

   The suggested document suite is:

   A Combination of Requirements and Release Plan documentation

   This document will serve the purpose of capturing the requirements and classifying them into releases. Release Planning and Project Planning can be performed here.

   Architecture and Design Documentation

   These documents will capture the design rationale and the architecture of the system.

   Test Documentation

   These documents will describe the test planning, definition of tests, and other testing related activities.

   User Documentation

   These documents must be delivered to the user along with the software product and include user and reference manuals, installation manuals and any other information that a user might require to operate and install the software product successfully.

   Templates for these documents can be used to facilitate their creation.

2. **Quality Assurance**

Suggested improvements deal with performing and managing product quality assurance. The improvements are mapped, to the SPICE processes, SUP.2 (Perform Configuration Management), SUP.3 (Perform Quality Assurance), SUP.4 (Perform Work Product Verification) SUP.5 (Perform Work Product Validation) and MAN.2 (Manage Quality).

A number of areas of improvement were identified during the assessment, but a few areas were targeted for immediate improvement based on the priority and risk. The three major issues we concentrate on in the quality assurance processes are:

(a) *Metrics* to help in measuring the process;

(b) *Reviews* to help in validating the work-products;

(c) *Traceability* to help in change management.

Metrics To provide better management of the quality of the product, a metrics program should be deployed. Metrics that provide information that helps in scheduling (time taken for any task) and process improvement (number of defects in each phase and during reviews) are good candidates for collection. To be effective, metrics collection should be semi-automated since the developers have little time to record extensive logs. As a result of the SPICE assessment, the recommended metrics to be collected are: number of defects detected during reviews and testing, number of class interface changes, time taken to do each task, severity of problems found during reviews and the reasons for the problems.

Reviews Reviews help in early detection of defects. Since defects are much more expensive to fix in the later stages of the project [Gra92], reviews improve the quality of the product and save time in the later stages of the project. Currently, the SEAF project is not performing any reviews. Reviews[BL89] should be performed at each phase of the project: requirements, design, code

and test. A checklist will be provided for each type of review to aid in the review process. Since a formal review is beyond the capability of a small project, we suggest that reviews be performed off-line. That is, the reviewers are notified of the availability of the work-product that is to be reviewed. The reviews can be performed at any time by the reviewers[WBM96a]. A form will be provided that records the results of the review. Once each reviewer is completed, their comments are posted to a newsgroup. The author can then read these messages and changes can be made accordingly.

Traceability As a first step towards a useful quality assurance program, traceability among work-products has to be established to demonstrate completeness [SC94]. Ideally, it should be possible to trace work-products back to requirements. To accomplish this, a view of all work products with their connections and relations to each other should be established. The traceability network is also effective in supporting change management, because work products affected by a specific change can easily be identified. When the reviews are being conducted for a work-product, the related work-products are specified, thereby establishing forward traceability. Defects discovered during testing should be traced back through the code, the design and then the requirements thereby establishing backward traceability. Since this information is stored in a database, reports of all the modules and their related documentation can be produced. Analyzing the traceability network helps in determining the root causes for defects. Root cause analysis aids in process improvement by identifying those phases and parts of the system that produced the most defects. Initially, the development team has to evaluate the information manually since automatic traceability analysis would be expensive to support. Therefore, reports will be provided, but the analysis is manual.

3. **Customer-Supplier relationship**

   Suggested improvements in this area are mapped to the SPICE processes CUS.2 (Manage Customer Needs) and CUS.5 (Provide Customer Service). The project does not face major difficulties in this area at its current stage. However, this

area was identified as a high risk area. Currently the project has one person acting as customer (the managing director), which will not be the case in the near future.

Customer change requests are managed in a semi-formal way. Typically, a new request is assessed with respect to its severity. Trivial requests are incorporated directly without any documentation, and the rest are queued in a waiting list based on their priority as assessed by the project manager.

Even though this process is acceptable at the current stage, it is doubtful that it can handle increasing numbers of customers. A new process has to be designed and enacted to meet the requirements of the anticipated situation. This process should be concerned with the time frame for handling requests and keeping customers informed.

All processes that were enacted by the process participants were defined as compulsory processes. All other applicable processes were defined as optional processes. For example, the team was already performing revision control and this was defined as a compulsory process. The team was not performing reviews and this was defined as an optional process.

3.5 Summary

An assessment of NewCo identified initial areas of improvement. Based on immediate needs, priority and risk, both the developers and assessors identified traceability, reviews and metrics as areas requiring immediate improvement. The remainder of the thesis focuses on the improvement opportunities and support in these three areas.

Chapter 4

The Quality Process and Quality Framework

This chapter introduces the quality process that is used as a basis of improvement and assessment activities for NewCo. Section 4.1 presents the quality process and Section 4.2 gives the desirable properties of the quality framework.

4.1 Quality Process

Implementing a quality system that satisfies the dynamic nature of a small project requires a quality model that can be tuned as rapidly as the development model. This model must be evolutionary to allow the introduction of new quality processes.

The Spiral Model[Boe88a] of the software lifecycle allows incremental evolution of the software development processes. It is generally accepted as the most realistic approach to developing complex software[Pre92]. We adopt and extend the traditional phases of the spiral model for process improvement as depicted in Figure 4.1[KVHS97].

4.1.1 Quality Spiral 0: Base-lining the process

In this quality spiral:

- The current process of the organization is modeled (formally or informally) to understand the type and nature of activities in the organization.
- The process is then assessed formally to quantify the starting point for the process improvement program, and to establish the maturity level of the current

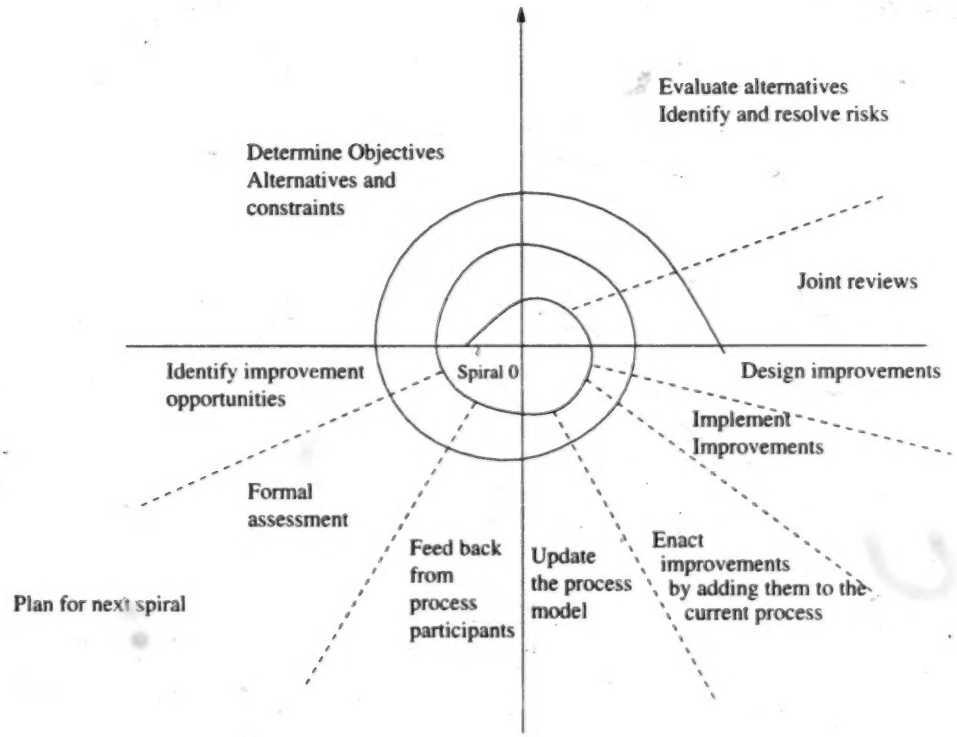

Figure 4.1: The Spiral Model for Process Improvement

process.

- Areas of improvements are then identified to furnish the starting point for the next spiral.

4.1.2 Quality Spiral 1-Onward: Process Evolution Cycle

Each spiral has four phases beginning with:

- Determine objectives for the current improvement cycle;
- Determine alternatives for implementing and enacting target improvements;
- Identify constraints imposed on each alternative.

The intent of the **First Phase** (upper left quadrant) of the spiral is to determine objectives for process improvement and identify constraints related to schedule budget and process enactment. Determining alternatives addresses process architecture, identification of best industrial practices to apply and plans for enactment of required improvements.

The objective of the **Second Phase** (upper right quadrant) of the spiral is to identify and resolve risks. As the traditional paradigm suggests, this phase includes:

- Identify areas of uncertainty and sources of risk;
- Evaluate alternatives relative to objectives, constraints and risks;
- Resolve major risk issues;
- Perform joint reviews with the process participants in order to resolve risks and agree on areas of improvement.

During this phase, the risk of process participants resisting the full enaction of the improvements has to be addressed. Joint reviews and the recruitment of a client representative for the development team are typical ways to resolve these risks.

The **Third Phase** (lower right quadrant) incorporates improvements in the current process. This phase includes:

- Tailoring the current processes, and/or developing new ones in order to incorporate the improvements in the current process model;
- Modifying the existing process model, testing the changes (the implementation step);
- Enacting the new/modified process.

In this phase the chosen alternatives are designed, implemented and enacted. Among alternatives to be considered are changing process standards, process programming, buying off-the-shelf software or introducing templates and forms.

The **Fourth Phase** (lower left quadrant) of the spiral is to plan for the next spiral. This phase includes:

- Collecting feedback from the development team;
- Formally assessing the new process after it has stabilized. Assessment results are used to quantify the current state of the process, to evaluate current spiral improvement efforts, and to suggest areas of improvements for next spiral.

4.1.3 NewCo Quality Spiral

The process assessment and improvement activities in NewCo are based on the quality spiral described above. Spiral 0 (Figure4.2) was the SPICE assessment performed on NewCo (described in Chapter 3). NewCo is currently performing Spiral 1. In the first phase, the objectives were identified as ISO 9001 certification and the constraints were the few resources available. In the second phase, the improvement opportunities were identified as a result of the SPICE assessment. The developers and assessors together decided to focus on traceability, reviews and metrics based on priority and risk. In the third phase, a quality system was designed and built in order to support these activities. NewCo must now use the system (beginning post-Beta) and incorporate changes that are a result of feedback from the participants. In the fourth spiral, NewCo has to perform another assessment to measure the effectiveness of the quality system and to identify other areas of improvement.

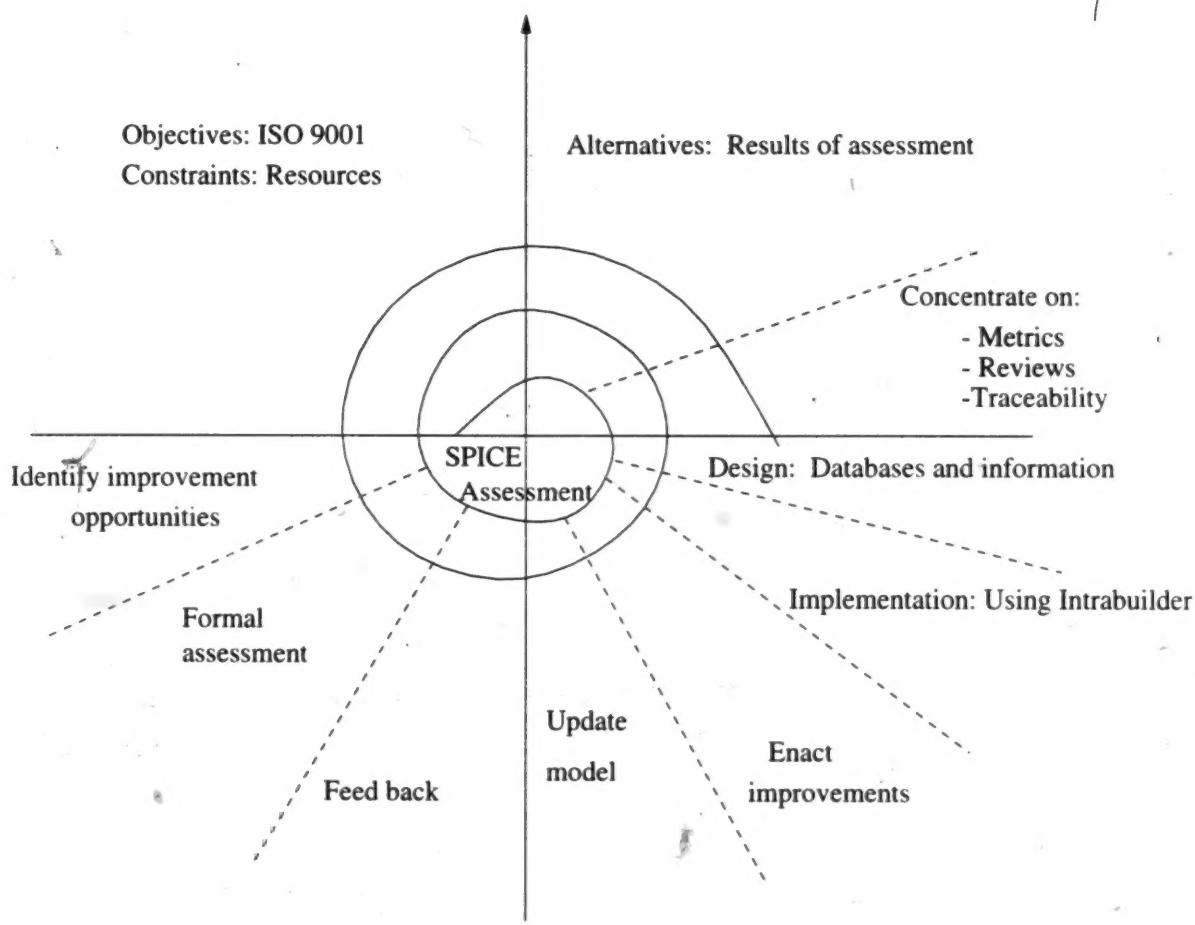

Figure 4.2: The Spiral Model for Process Improvement for NewCo

4.2 Quality Framework

As mentioned earlier, since a small software organization does not have many resources, it needs a quality system that is effective, affordable, scalable and usable. We propose a quality framework that addresses these needs. The quality framework is based on the quality process described above. The quality framework provides support to perform the quality activities and addresses the resource constraints faced by small software organizations. The goals of the quality framework are to:

- provide a repository of processes to choose from; and directions on how to use the repository;
- automate some of the tasks to be performed;
- help a small software organization not only improve its process but also become ISO 9001 compatible.
- help a small software organization out-source its quality assessment and improvement activities;

4.3 Properties of the Quality Framework

A quality framework provides support for performing quality processes. The nature of the support should satisfy the properties: effective, affordable, scalable and usable. But the actual processes that are performed are the ones that evolve.

A quality framework can not only help a small software organization improve its process and product quality, but also help it in becoming ISO 9001 certifiable. It should support out-sourcing of quality and provide automated support.

Each of the properties are discussed in detail below.

Effective

The effectiveness of the quality system (that is generated using the quality framework) can be measured during a formal assessment such as one described elsewhere in this thesis (Chapter 3). If the organization has used the system and performed the recommended activities, a comparison of the results of assessment before and after the use of the quality system will provide a measure

of the effectiveness of the system. The assessment should also identify other improvement activities that have to be incorporated in the quality system. The assessment activities may be performed as part of a spiral process improvement model.

Scalable

The quality framework should provide support for the quality activities of an organization at each stage of the product development. In addition, it should also provide support for the processes to evolve with the development process and be extendable when the organization changes. That is, the quality system must grow or shrink to meet the needs of the organization. For example, for an organization that is introducing a quality assessment and improvement program, the framework will consist of only a few core components. These components will evolve as the organization's processes mature.

An organization may expand in a number of directions:

1. *People*

   As an organization expands, it loses some of the properties that are special to small organizations (see Chapter 2). For example, as the number of people increase, it is difficult for all the team members to keep track of what every other team member is doing. Therefore, additional processes (such as walk-throughs) that help to keep team members informed about the project will be needed. The quality framework must provide support for these activities.

   Small organizations generally have a flat organizational structure, but as the organization expands, the structure will become hierarchical. Therefore the communications which are generally informal (via phone or e-mail) in small organizations, have to be formalized. For example, documentation in a small organization is typically informal since each team member knows what the others are doing. But as the organization expands, documentation has to be more structured and organized. The quality framework could provide document templates in order to make this task easier.

2. *Process Maturity*

   Since one of the goals of the quality framework is process improvement, the framework should support changes to an organization's processes as they mature. For example, for an organization that is introducing elements of a quality system, reviews and collecting a few basic measurements may typically comprise of the first spiral. In the second spiral, more measurements and other activities such as training may be identified for improvement. The quality framework should support this change without substantial overhead.

3. *Projects*

   The organization may accept more contracts and work on more than one project concurrently. The quality framework should then scale up in order to handle more than one project. Some activities can be defined organization wide, whereas some will be special to each project. The quality framework should allow this customization.

Affordable

The quality framework must make quality affordable for small organizations since they have very few resources. A small organization typically cannot afford the startup costs for a quality system such as selecting and customizing quality assurance processes. The quality framework can reduce this startup time.

A typical small organization first starts product development and begins the quality activities toward the end of the development cycle (typically testing). This means that the organization has to ultimately spend more time fixing defects that could have been caught early in development. This scenario is probable because a small organization does not have the resources to select and customize quality processes. For example, in order to perform reviews during development, a number of support activities have to be performed. The types of reviews to be performed have to be selected and the measurements to be collected should be decided upon. Then, forms for recording these measurements have to be created. The storage of the measurements (in a spreadsheet, database or on paper) has to be planned and performed, and the generation of

reports facilitated. All this takes time that could be spent in development and therefore is not performed.

The quality framework helps reduce the startup time because the activities (and their procedures) to be performed are specified. In order to further reduce the cost, forms and templates are provided for most of the activities and generation of reports is automated. This means that the effort expended at the beginning of the project in order to customize the quality framework to the current project is significantly reduced. The quality activities are performed parallel to the development activities from project inception.

Usable

The quality framework should be easy to use and learn. As the framework is used, usability tests can provide valuable feedback about how information should be presented and collected.

4.4 Defining a Quality Framework

In order to help a small software organization out-source its quality assessment and improvement activities, the interfaces between the quality and development activities should be well defined. Most of the activities in a small software development organization are development activities. Quality activities are not performed because of the drain on the resources.

When a new project starts, a process from the pool of generic processes is selected.The pool contains processes that are described in the literature and/or based on previous experience. The generic process may be customized to suit the needs of the project. The processes that are selected include both development processes and quality assurance processes (see Figure 4.3). When these selected processes are enacted, they become actual processes. Development processes are those that directly specify, implement or maintain a product[KVHS97]. Quality assurance processes are employed during a project life-cycle to reduce defects, measure process efficiency, estimate product quality and perform other tasks to ensure the quality of the product. The actual development and quality processes interact through the following set of

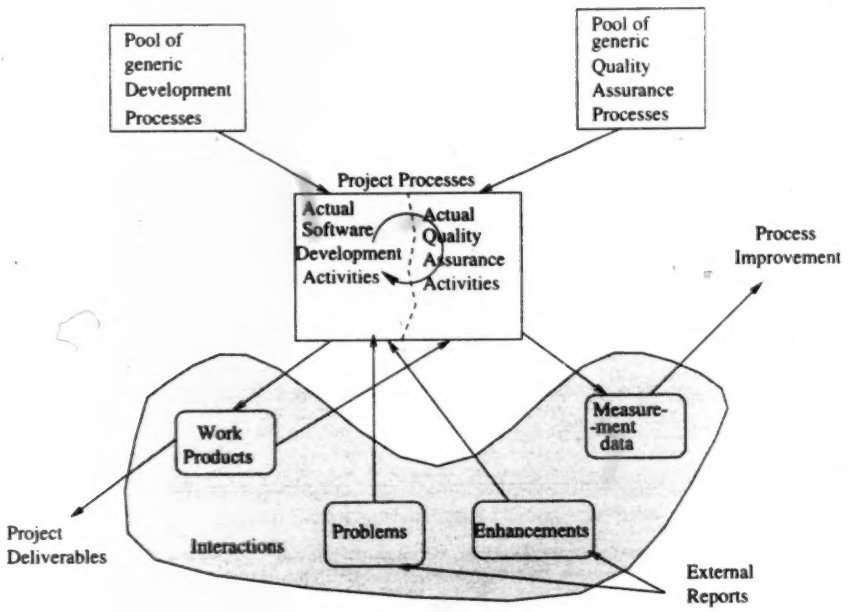

Figure 4.3: A model for interactions between Development and Quality processes

repositories:

Work-Products are the various outcomes of the development processes. Work-products include both project deliverables and internal items. Each work-product should be fed into at least one other actual process.

Problems are reported as a result of failures uncovered in the quality assurance process, including external reports from customers. They are mismatches between expected and actual behavior.

Enhancements are recorded as external reports that focus on things such as suggested enhancements and requirement changes, and do not include specific malfunctions in the software.

Measurement Data include product and process related measurements. This data is used as a basis for process improvement.

Work-products are produced as a result of development processes. Quality activities such as testing or reviews performed on them result in measurement data. This

data is analysed and used in process improvement.

The interfaces between the quality and development activities are not clearly defined in a small software organization because the developers typically do those quality activities that are performed. Figure 4.4 shows the inputs, outputs and states of each process.

The actual processes are divided into compulsory and optional processes. Compulsory processes must be performed as specified, while optional processes can be partially enacted, depending on the needs and requirements of the project at various points in the cycle. In a small organization, the set of compulsory processes consist of the basic development activities and most of the quality assurance activities are informal and will initially be viewed as optional.

In a quality system that is affordable, any process or activity can be compulsory or optional depending on the project context. For example, *develop detailed design* should be a compulsory process for large projects, but it can be optional for small size projects. Similarly, walk-throughs may be omitted during the alpha phase, but instantiated during beta. The benefits and additional costs of moving a process to the compulsory side, as well as the risks of moving it to the optional side must be addressed.

The quality assessment and improvement activities of an organization can be out-sourced since these are supporting activities to software development. This will help a small organization to focus on software development. When a small organization out-sources quality, the quality framework defines all the support needed to perform quality activities. Out-sourcing quality allows an organization to use customized forms and templates that are based on the experience of other organizations. By performing the specified quality activities, the organization can improve its processes and become ISO 9001 certifiable.

4.4.1 Automated Support

The quality framework should provide automated or semi-automated support to the development team for performing quality activities. Forms that help in collecting information about reviews, provide document templates, or support metrics collection are a few examples. Generation of reports automatically will also provide valuable

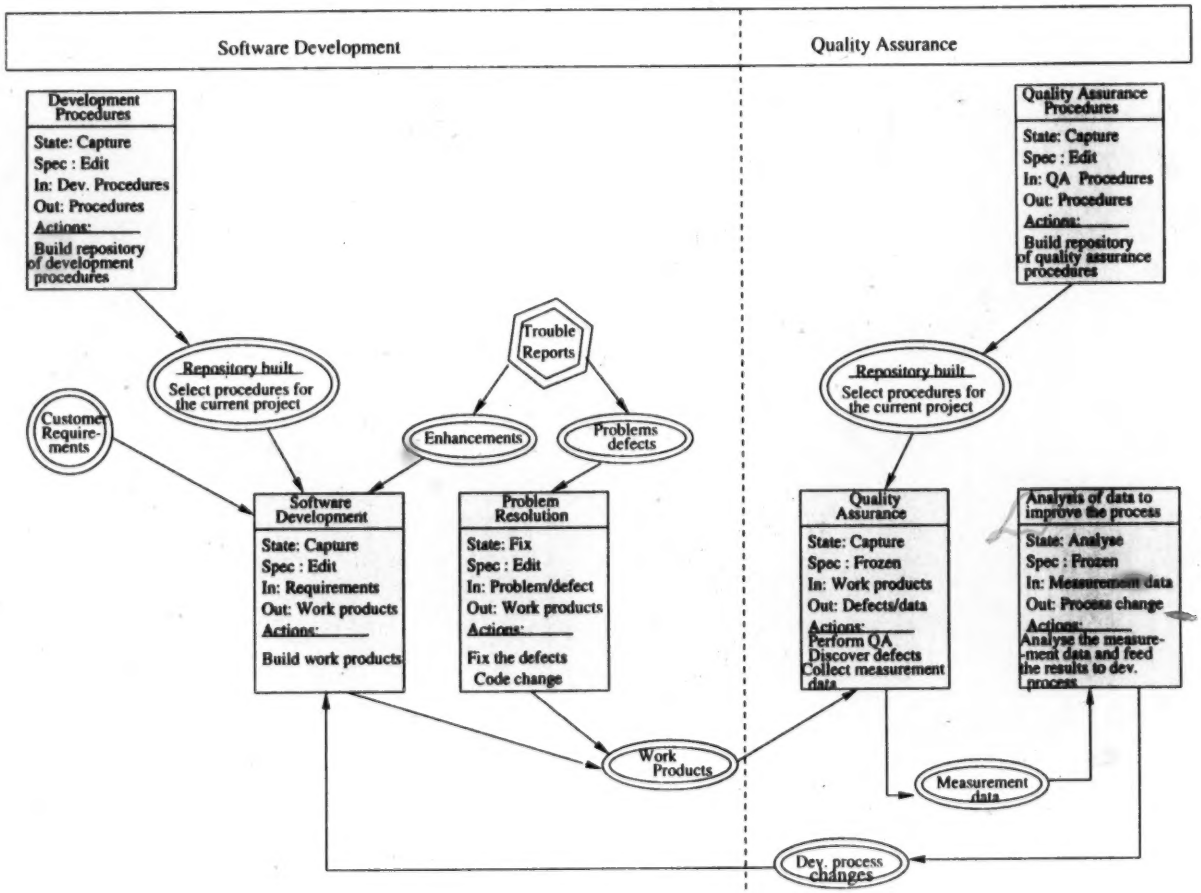

Figure 4.4: Work-product interaction between development and quality processes

information for analysis. The current process can be benchmarked (i.e. compared) with historical information of the organization or with information from a similar organization.

4.5 Summary

The spiral model has been proposed for process improvement to help an organization plan and implement their quality assurance activities in stages. Since the spiral model assesses the risk at each phase, the viability of the quality program at the end of each spiral can also be measured.

The proposed quality framework provides affordable, effective, scalable and usable support for a small software organization to perform quality activities. The quality framework incorporates the spiral model in order to evolve as per the needs of the organization. The quality framework provides a means of assisting a small software organization out-source its quality assessment and improvement activities and providing automated support.

Chapter 5

Components of the Quality System

The quality framework just outlined will be used as a basis for developing the quality system for NewCo that has three basic components as derived from the SPICE assessment:

- Reviews
- Metrics
- Interpretation of quality information

The quality framework provides the support to perform quality activities, but it is the actual processes that are performed that evolve and change. For example, a review process that is suggested for a small organization should be effective, affordable, scalable and usable. That is, the review process should evolve with the organization's needs.

5.1 Reviews

Reviews have been classified in numerous ways. In this thesis, reviews are classified as [Hum95, Fre90, SC94]:

1. *Walk-throughs* are used to educate the participants as well as to find defects.
2. *Inspections* follow a well defined formal process and are aimed solely at detecting defects.
3. *Formal and Informal Technical reviews* are used in order to find defects and to discuss their solutions.

All the above types of reviews can be applied to any type of work-product (for example, requirements, design, code, test, documentation etc.) generated in any phase of the project. Reviews have to be held in any organization that wants to improve its process. The kinds of reviews used may be different as the organization expands and evolves. For example, in an organization that is introducing reviews, the review process may be informal(walk-throughs are an example); but as the organization expands and its processes become more mature, the process may be formalized.

A software review is an evaluation of a work-product to detect differences from planned results and to recommend improvements [SC94]. In addition to validating work-products, they help in [Fre90, SE93, Wei91] the following aspects:

Communication

Reviews help to communicate technical information thereby replacing or supplementing formal written documentation. Review participants learn about languages, tools, techniques and the work-product at hand. The participants learn how much they know compared to others and how much more they have to learn in an environment that is non-threatening.

Improving schedule performance and eliminating redundant work

Reviews help in early removal of defects which means that project time is not spent in designing, documenting and implementing something that does not meet the requirements. If design faults are found before coding is done, or if mismatches between the specifications and product are caught early, much of coding and recoding is eliminated.

Confirming parts of the product

Reviews are not an alternative to testing. Testing is performed on code, but reviews can be performed on any work-product; for example, a design document. Therefore, reviews catch faults directly and early on, rather than waiting for the fault to show in testing when it will be expensive to fix. Reviews are also a method of verifying that the product conforms to the requirements, and that it has no faults. Performing reviews at every stage in project development helps to achieve technical work that is uniform in quality. Reviews help in resolving issues of design and development.

5.1.1 Walk-throughs

IEEE STD 1028-1988 [ANS89] defines the requirements for a walk-through and its objectives as primarily *to find defects, omissions and contradictions and to improve the software element, and to consider alternative implementations.* Other objectives include *exchange of techniques and style variations and education of the participants. A walk-through may point out efficiency and readability problems in the code, modularity in the design or untestable design specifications.*

Walk-throughs [Fre90, WBM96a] can be held at any stage of the project and generally start with a presentation. Though the objective of the walk-through is to find defects, the participants are permitted to make suggestions for improvement. Walk-throughs, by their nature, ensure that a large amount of material is covered in a short time. When the project team is large, this is an excellent way to bring the participants up to date. Prior preparation of the presentation material or work-product under review is not essential.

Structured walk-throughs [You89] have been defined as a group peer review of a product. The reviewers have to find defects, but suggestions are permitted.

Since advance preparation is not required for a walk-through, each participant will have a different depth of understanding. This may make the walk-through less productive than an inspection. Since the participants are allowed to digress from finding errors into discussing improvements, the rate of finding errors may not be very high.

Walk-throughs can be performed during combined reviews with the customer. They help in clearing up any miscommunication or misunderstandings. Design and code walk-throughs can be used in order to communicate system components to other designers and programmers.

5.1.2 Inspections

Inspections are a means of verifying work-products where small groups of peers manually examine them to ensure that they are correct and conform to product specifications [SE93]. The main purpose of inspections are to detect defects. No attempt is made to solve the defects or provide suggestions for improvements during inspections.

The difference between inspections and other forms of quality control such as reviews and walk-throughs is the formality of the inspection process. A set of specified steps are defined and have to be followed. An inspection process requires several roles: the moderator, a reader, a recorder and one or two inspectors. There are a number of variations of inspections but most of them follow the six principal steps [WBM96b] defined in Fagan's inspection process [Fag76]:

1. *Planning:* When a work-product is completed and is ready to be inspected, an inspection team is formed and the moderator is designated who ensures that the work-product meets the entry criteria. Roles are assigned to the inspectors and copies of the work-products are distributed.

2. *Overview:* This optional step may be necessary if the inspectors are not familiar with the development project.

3. *Preparation:* Inspectors prepare individually by studying the work-product and related materials. Checklists may be used to help detect common errors. IEEE STD 1028-1988 suggests 1.5 hours per inspector preparation time; but more time may be needed if the inspectors are unfamiliar with the project.

4. *Examination:* This is the meeting where the work-product is reviewed. The moderator ensures that the inspectors are sufficiently prepared and the reader presents the work-product. Then all inspectors look for defects. No attempt is made to find solutions for the defect and all criticism of the author is avoided. Examination length is generally limited to a maximum of two hours because the inspectors ability to discover defects decreases rapidly [Fag76]. At the end of the meeting, the team decides if the work-product should be accepted as is, re-worked with moderator verifying the results, or reworked and re-inspected.

5. *Rework:* The defects are corrected by the author after the examination meeting.

6. *Follow-up:* The moderator checks the author's corrections, and if satisfied, the inspection is closed and the work-product placed under configuration management.

Design and code inspections are necessary in order to find defects and ensure that they meet the specifications. Estimates of between 50 to 75% of all design errors can be found with inspections [Gra92]. Fagan's inspection process [Fag76] is a well-defined formal process for finding defects in design and code. Since Fagan proposed this approach in 1976, a number of variations have been created and inspections have been successfully used in the industry [Rus91, Wel93, GS94, Fow86, BB91, GHP86]. Most inspections require an examination meeting where defects are found. But in large projects or projects where the team members are at geographically diverse locations, this is difficult and a lot of time is wasted in trying to schedule the meeting. Studies have shown that 20% of time between requirements and starting design is spent waiting for reviewers to meet [BPV93]. A number of solutions have been proposed to solve this problem. Some of them are discussed here.

A distributed, collaborative software inspection method [MDTR93] supports a structured meeting and lets participants work from different physical locations. Inspection material is present on-line and inspection records are created during inspections enabling review metrics collection. A tool called CSI (Collaborative Software Inspection) provides support for synchronous (discussion and categorization of faults) and asynchronous (individual reviews to discover faults) activities. CSI allows annotation of the work-product asynchronously and synchronously by creating hyperlinks between the document and reviewer's annotations. Discussion is supported by a teleconferencing tool called Teleconf which helps in recording the results of the inspection.

A computer supported cooperative work environment called CSRS (Collaborative Software Review System) was designed to aid in performing formal, technical reviews [Joh94]. The FTArm (Formal, Technical, Asynchronous review method) is a computer mediated method that addresses the problems that are associated with an inspection meeting such as insufficient preparation by the participants, moderator domination, recording difficulties and clerical overhead. The FTArm process has seven stages that are not dependent on the work-product or the development phase. CSRS stores in a database the outcome of the inspection.

Inspections, as defined by Fagan, have between three and six participants, with no management personnel. For a small organization which does not have access to a larger group, having six participants is difficult. Therefore, a two-person inspection

method has been proposed [BL89] which benefits the less experienced programmers. The two-person method is useful when the organizations do not have or are reluctant to assign resources for inspections.

5.1.3 Technical Reviews

Design and code must be reviewed separately [Hum95] in order to save implementation time and to make sure that the design is of good quality before doing the coding. But the same techniques can be applied to both of them. Design and code must be reviewed [Hum95, Fre90] in order to ensure that they meet the specifications and can be implemented.

Informal reviews may be conducted between two or more people when they review each others work. Since the review is informal, there are no rules and regulations and the results are not reported to anyone other than the owner of the work-product.

Desk-checking is another form of review where a single person reviews the work-product. The success of this approach is totally dependent on the experience of the reviewer.

Formal design and code reviews should be performed on the most critical parts of the system [Boo95]. The system may be reviewed in parts, and the parts that have been approved should be coded.

Some of the strategies for design and code reviews are discussed below. Most of the strategies review the design and code in stages [Hum95].

Round-Robin reviews are useful when all the participants are at the same level of expertise. In this type of review, the participants take turns reviewing a portion of the work-product. This ensures that every participant learns about the work-product thoroughly. A number of variations of this review type are in use to educate the programmers about the system at hand [Fre90]. The participants can discuss solutions for any faults that are found.

Selected Aspect reviews [WBM96a] are those that concentrate on a pre-selected set of aspects of the work-product. Specific items to look for are typically specified as a checklist. Selected aspect reviews can be performed as a series of reviews where different aspects of the work-product are reviewed.

Phased inspections [KM93] are another variation of selected aspect reviews, where each phase concentrates on one or a small set of related properties. These inspections may be performed by a single inspector or by multiple inspectors. In a single-inspector inspection, one person checks the work-product based on a specific aspect. In multiple-inspector inspections, a number (more than one) of reviewers inspect the work-product individually and then reconcile the defects later.

Active design reviews [PW87] are a type of selected aspect reviews where the designers ask the reviewers questions through questionnaires. The designers prepare questionnaires on the work-products to be reviewed. The reviewers answer them and may have to provide justification for accepting/rejecting the design. This type of review helps in focusing the attention of the reviewers on specific parts of the work-product. Overall active design reviews are also held to find any errors that might have been missed in the specific reviews. Each review has three stages. The first is a brief overview of the module presented to the reviewers; and the reviewers are assigned to document sections and reviews. In the second stage, the reviewers review and meet the designers to resolve any questions. Finally, the designers read the questionnaire results and meet with the reviewers to resolve any questions.

Test Plans have to be reviewed before coding begins [Fre90] to make sure that tests do not just test the code. Test plans have to identify, among other things, test inputs and expected outputs. Walk-throughs or technical reviews may be performed.

Release review takes place between releases and re-evaluates the system boundary and context, identifies major extensions or deletions and designs for the new release in order to assess its impact on the existing architecture.

All documentation must be reviewed at least once [Fre90] in order to ensure that it is technically correct. Code comments, design documents, requirements documents are usually reviewed during their respective reviews. But any user documentation must be reviewed separately in joint reviews with the user. These reviews may be in the form of walk-throughs or technical reviews.

5.2 Metrics

Software metrics help in measuring the effectiveness of the process and in estimating and scheduling for future projects. Software metrics have been defined in a number of ways [Mil88, Cd97], but in this thesis we use Goodman's [Goo93] definition:

Software metrics are the continuous application of measurement-based techniques to the software development process and the products to supply meaningful and timely management information, together with the use of those techniques to improve process and its products.

As the organization grows, the types of metrics collected will change. Since NewCo is introducing a quality program, the metrics that are collected are review, inspection and work-product related metrics. As the organization's processes mature, the metrics collected will be different; metrics relating to code, efficiency of a development team etc. might be collected.

Software metrics can therefore be used in order to improve all aspects of the software development process and its management. They can also be used to predict and estimate the cost (in terms of time and money) of the project and any overruns.

The goal of software metrics, among other things is [Cd97]:

- Measurement of development activities to predict or estimate subsequent development costs
- Measurement of quality activities in order to predict or estimate subsequent development costs to achieve acceptable product quality
- Measurement of the development and quality work-products in order to measure the efficiency of the process
- Measurement of the code to measure the quality of implementation

A quality assurance system is being introduced in NewCo. Very little is available in the literature describing experiences in the introduction of quality assurance systems. Some suggestions are [Wal94] to introduce a basic system first in order to test the viability of the system. The basic system should introduce the minimum activities

necessary for the quality assurance program. In the specific case of NewCo, support is provided for collecting information pertaining to reviews and inspections in the first development spiral (See Chapter 4).

5.2.1 Collection of measurements

A reviewed work-product is complete, correct and dependable to build upon. Reviews are therefore a reliable way of measuring project progress. Review reports provide information about the effectiveness of the review process, number and type of defects, action items, defect-prone components and development performance [SE93].

Research has shown that user documentation and test plans are not inspected, design and code inspections occur late in their phase and a majority of inspections are held in the latter half of the software life-cycle [Shi92]. For inspections to be effective, they have to be distributed evenly across the software life-cycle.

A tool has been prototyped in order to aid in the collection and reporting of review information. This information is stored in a database. Although most of the review information collected is conformant with the industry practice [Fre90, SE93, BB91, Wel93] it has been tailored to suit NewCo's development environment.

Work-Product Identification Data is used to identify each work-product and its related work-products. This facilitates traceability [SC94]. All information pertaining to a work-product is collected. This information will help in identifying work-products that are related to the one currently under review and hence the ones that might need revising due to changes. The specific information collected is:

- Project name
- Work-product name
- Work-product status (New, Tested or Modified)
- Related work-products
- Type of work-product (Document, Code)
- Kind of work-product (Architecture Classes, File, Document, Business Classes)

- Author of work-product
- Size of the work-product

Review Information Data provides information about the review. This information helps in controlling the process and predicting the length of the review. This information can be used to create schedules for future projects.

- Reviewer (or moderator) Name
- Date of the meeting
- Review type (requirements, code, design, release, test)
- Preparation time (hours)
- Examination time (hours)
- Estimated rework
- Actual rework
- Number of inspectors
- Work-product disposition (accept, conditionally accept, reject)
- Meeting type (for inspections only : inspection, re-inspection, overview)

Product defect data provides data about each defect that was discovered in the reviews. For a technical review (not an inspection), the reason for the defects are also collected [Wal94] which helps in determining the major causes of defects. Since inspections are geared towards finding defects and not the causes [Fag76], this information is not collected for inspections.

- Location of the defect (line number)
- Description of the defect
- Defect type (requirements, change report, review report, trouble report)
- Defect Class (wrong, missing, extra)
- Severity (medium, major)
- Defect status (Open, Closed, Next Release)
- Reason for the defect (technical reviews only)

5.3 Interpretation of quality information

The quality system will enable collection of metrics and will generate reports for analysis. But how can an organization measure how effective the program is, or how to use the metrics to improve the process? In order to do this, the organization has to have a yardstick or measurement basis by which to compare the measurements. For example, if in requirements reviews, an organization finds 10 defects, is this good or bad? If the organization has historic data that shows that requirements reviews detected more than 50 defects, it means that there may be something wrong with the current review process. If there is no historical data, the organization needs to look at similar metrics collected for similar organizations. There are a number of studies that give estimates for large companies [Gra92], but none as yet that do the same for small companies.

The analysis of the quality information has to be performed by the team members. The quality system will provide reports, company profiles and other relevant information. In order to help in comparing and analysing the quality information, company profiles may be used. This profile will be automatically generated at the end of each stage of development. The company profile will look like Figure 5.1. This profile gives an overview of the activities in the organization.

Team size and development strategy are useful metrics when comparing two project or organization profiles. Profiles of larger organizations will be different because they have more resources in terms of time, people and money. Large organizations also generally have a separate quality assurance group that makes sure that the quality activities are being performed. Small organizations do not have the resources for a separate quality assurance group and so each team member has to be self-motivated in order to perform quality activities.

The time spent in each of the phases is another useful metric. Averages from 125 Hewlett-Packard [Gra92] projects show that projects invest about 18% of total effort during specifications/requirements phase, 19% during design, 34% during coding and 29% during testing. These figures act as a sanity check for project estimates and help in future scheduling. This data is based on large projects and could be different for small organizations.

Company Profile for NewCo

Team Size: 5

Development Strategy: Evolutionary prototyping

Reviews

	Defects	Avg. Time	Avg. Resources
Requirements	3	2 hrs	2 people
Design	10	20 hrs	3 people
Code	50	30 hrs	2 people

Inspections

	Defects	Avg. Time	Avg. Resources
Requirements	5	2 hrs	3 people
Design	20	20 hrs	3 people
Code	50	20 hrs	3 people

Avg. defects found in testing: 100

Avg. defects found by customers: 20

Time spent in req. phase: 200 hrs

Time spent in design phase: 300 hrs

Time spent in implementation phase: 400 hrs

Figure 5.1: Sample Company Profile

The resources used (time and people) versus the defects found in reviews and inspections provide valuable data about the efficiency of the review and inspection process. This data varies depending on the experience of the reviewers and inspectors. Historical data from the same organization would provide valid information for comparison.

5.4 Summary

The Quality Framework addresses the quality needs of a small software organization by providing support for the quality activities. The quality framework is scalable, affordable, out-sourcable, effective, usable and is supported with automated tools for the quality activities. The quality system for NewCo has review, metrics, traceability and interpretation of quality information as its core components.

Chapter 6

Design and Implementation of the Quality System

This chapter describes the design and implementation of the quality system that was prototyped to provide support for performing the improvement opportunities identified during the assessment. Joint reviews between the developers and the assessment team identified traceability, reviews and metrics as immediate areas on which to concentrate.

6.1 Requirements of the Quality System

The goal of the Quality System is to provide support for NewCo to perform quality activities that are part of a process improvement program that will allow NewCo to achieve ISO 9001 certification. The quality activities that are to be performed by NewCo have been identified by the SPICE assessment. Based on the assessment, the following requirements for the Quality System have been identified:

1. Review mechanism

Two types of reviews will be supported by the system:

- Technical reviews that occur between two or more members. They should allow participants to express ideas and consider alternate solutions. These reviews seek to resolve issues whenever possible.
- Inspections are a means of verifying products by manually examining them to ensure correctness and conformance to specifications. Inspections are

aimed at discovering defects and not at resolving any of the issues that are discovered. Issues arising from inspections are resolved after the inspection meeting.

2. **Metrics and information collection mechanism**

   A system that enables the collection of metrics and other relevant information in an easy manner.

3. **Problem tracking mechanism**

   A system that records the problems that are discovered during technical reviews and inspections and their status. When any of these problems are solved, the change in status is recorded in a database.

4. **Forward and backward traceability**

   The system establishes traceability by collecting information about the documents/code related to the document/code under review. Each problem is tracked individually. Reasons for the causes of the problems are also recorded.

5. **Report Generation**

   A number of reports which summarize the metrics and information collected during technical reviews and inspections will be generated.

6. **Conforming with the Quality Framework**

   The quality system should conform to the properties of the quality framework. This prototype implements the core components of the quality system which supports reviews, metrics and reports for interpretation of quality information.

6.2 Architecture

The Quality System is designed to be tightly coupled with Cafe Seaf and yet can be disabled when not needed. The responsibility of performing reviews lies with the development team. The quality system is user-driven, in the sense that it does not force the development team to perform reviews.

Intrabuilder[1] was used (see Appendix B for the technology review) to develop the

[1] Intrabuilder is a trademark of Borland Corporation

quality system because it provides easy and efficient means of creating web-based applications as well as a database backend connected to an HTML frontend [Kru].

6.2.1 Database Design

The Database server accesses the local tables in order to retrieve/store information. An Entity-Relationship diagram of the tables is shown (see Figure 6.1). Each review and inspection must be associated with a work-product. Each problem must be associated with a work-product and each reason must be associated with a problem.

Problems Table

The Problems Table is used to store the problem-specific information. See Table 6.1 for an explanation of the fields.

Name	Reason
Work-product Name	The name of the work-product
Location of defect	The line number the defect was found on
Description of the defect	The description of the defect
Defect Class	One of: Wrong, missing or extra
Severity	One of: Medium, Major
Name of the problem	A unique name that describes the problem
Type of problem	One of: Change request, trouble report, review report or requirements
Kind of review	One of Inspection or Informal Technical Review
Status of the problem	One of: Open, Closed, Unsolvable or Next release

Table 6.1: Problem Table

Reasons Table

The Reasons Table is used to store the root causes for the problems. See Table 6.2 for an explanation of the fields.

Work-Product Table

The work-product table is used to store the information specific to work-products. See Table 6.3 for an explanation of the fields.

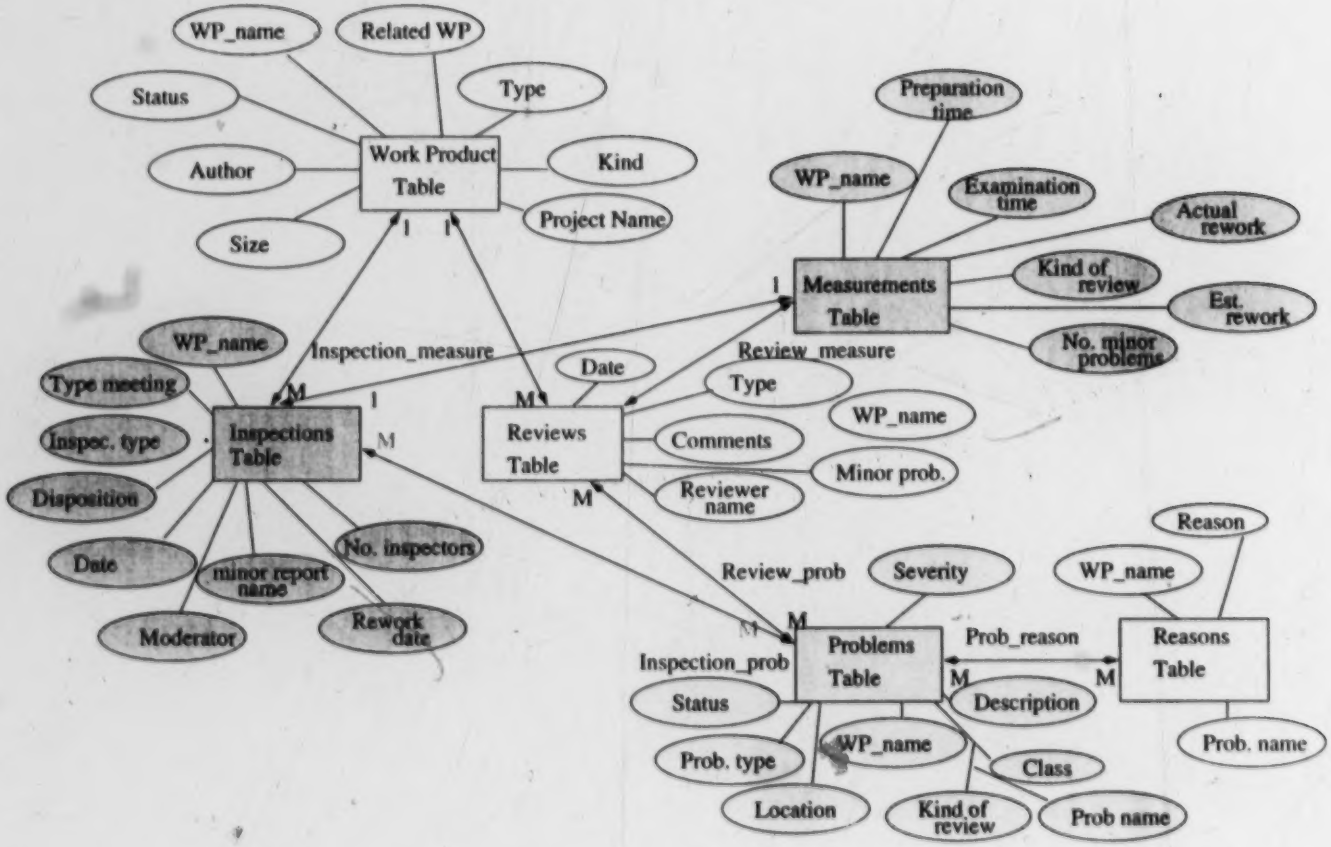

Figure 6.1: Entity-Relationship Diagram

Name	Reason
Work-product Name	The name of the work-product
Name of the problem	A unique name that describes the problem
Reason	The reason why the problem occured

Table 6.2: Reason Table

Name	Reason
Work-Product Name	The name of work-product for which the review is being held
Related work-products	All the work-products related to the current one. This establishes forward traceability
Type of work-product	One of: Code, Document
Kind of work-product	One of: Business Classes, Architecture Classes, Document, File
Work-product status	One of: New, tested, modified
Size of work-product	Size
Project Name	The project to which the work-product belongs
Author	The author of the work-product

Table 6.3: Work-product Table

Inspection Table

The inspection table is used to store all information specific to inspections. Table 6.4 gives an explanation of the fields in the table.

Review Table

The review table is used to store information specific to reviews. Table 6.5 gives an explanation of the fields.

Measurements Table

The measurements table is used to store the measurement data that is collected. Table 6.6 gives an explanation of the fields.

6.2.2 Reports

Measurements serve a number of functions [Gra92]. They help in estimating and tracking project progress, achieving a desired level of quality, analyzing defects and process improvement. But the measurements collected in each organization reflect the

Name	Reason
Work-product Name	Name of work-product
Type of inspection	One of: Requirements, Code, Test, Release, Design
Moderator	The name of the moderator of the inspection
Meeting Type	One of: Inspection, Re-inspection and Overview
Date of the meeting	The date inspection took place
Number of inspectors	Number of inspectors
Rework completion date	The deadline for completing the rework
Work product disposition	One of: Accept, conditionally accept, re-inspect
Report Name	The name of the file where the minor defects are specified.

Table 6.4: Inspection Table

Name	Reason
Work-product Name	Name of work-product
Type of review	One of: Requirements, Code, Test, Release, Design
Date of review	Date the review was completed
Comments	Any short comments on the document under review
Reviewer Name	The name of the reviewer
Minor problems file	A file where all the minor problems are reported. All other types of problems will be tracked individually

Table 6.5: Review Table

Name	Reason
Work-product name	Name of the work-product
Preparation time	How long the inspectors had to prepare (in hours)
Examination Time	How long the inspection lasted (in hours)
Number of minor problems	The number of minor problems in the work-product
Kind of review	One of Inspection or Review
Estimated rework	Estimation of how long rework would take (in hours)
Actual rework	How long the rework actually took (in hours)

Table 6.6: Measurements Table

state of that organization and cannot be transferred to another. Historical databases have to be built for each organization and each project individually.

The information collected in the tables has to be presented to the team members in an easy-to-read format. Queries have been grouped into a number of categories based on their function.

1. **Process Improvement Queries** are those that help in measuring the effectiveness of the process. These reports aid in making decisions about the need for changes in the process.

   (a) *What is the reason for the most number of defects?* (Incorrect understanding of requirements, incorrect requirements etc..)

   This query will help in finding and eliminating the cause for the most number of errors. For example, if the most number of errors are caused by *Incorrect understanding of requirements*, then it probably means that more time has to be spent in the requirements phase in order to understand them. Since there is evidence that requirements defects are about 100 times more expensive to fix [Gra92] in the later stages of the project, it is well worth the extra time spent in this phase.

```
SELECT problem."WP_name" , problem."Severity" ,    reason."Reason" ,
(count( reason.Reason )  )  as Total_wp
FROM "problem.db" problem , "reason.db" reason
WHERE ( problem.WP_name = reason.WP_name )
GROUP BY problem. "Severity" , problem."WP_name" , reason."Reason"
ORDER BY Reason
```

   (b) *How many defects were discovered during reviews in each phase?*

   This report provides information about how many defects were discovered during reviews in each phase. There are several ways this report can be used. For example, if there is a large disparity in the number of defects detected amongst phases, then the effectiveness of the review process and review participants in certain phases should be checked.

```
SELECT review."Type" , problem."Severity" ,    review."WP_name"
```

```
FROM "review.db" review , "problem.db" problem
WHERE ( problem.WP_name = review.WP_name )
GROUP BY   problem."Severity" , review."Type" , review."WP_name"
ORDER BY "review.db"."Type"
```

(c) *How many defects were discovered in inspections in each phase?*

This report provides information about the number of defects found during inspections in each phase. This information helps in determining the effectiveness of the inspection process.

```
SELECT inspection."Type_inspec" , inspection."WP_name" ,
problem."Severity"
FROM "inspection.db" inspection , "problem.db" problem
WHERE ( inspection.WP_name = problem.WP_name )
GROUP BY inspection."Type_inspec", inspection."WP_name",
problem."Severity"
ORDER BY Type_inspec
```

(d) *Information about all the problems.*

This report provides information about all the problems in the system and classifies them based type of review performed (technical review or inspection).

```
SELECT * FROM "problem.db" ORDER BY Kind_of_review
```

2. **Status Queries** are those that provide status information about the problems discovered during technical reviews and inspections. This report shows the problems that are closed, those that are open and those chosen to resolve in the next release. This report helps in tracing problems that should have been solved and have not.

   (a) *The status of all the problems.*

   ```
   SELECT * FROM "problem.db" ORDER BY WP_name
   ```

3. **Traceability Queries** are queries that help in tracing work-products.

(a) *The work-products that have been reviewed and their related documents.*

This report displays the work-products and their related work-products.

```
SELECT problem."WP_name" , workp."Rel_WP" , problem."Status" ,
workp."Author" , workp."Type"
FROM "problem.db" problem , "workp.db" workp
WHERE (problem.WP_name = workp.WP_name )
GROUP BY  problem."Status" , workp."Author" , workp."Type" ,
problem."WP_name" , workp."Rel_WP"
ORDER BY WP_name
```

(b) *Information about all the work-products.*

This report displays all information about all the work-products.

```
SELECT * FROM "workp.db" ORDER BY WP_name
```

4. **Management Queries** are queries that help in scheduling. They are reports that provide information about how long the team and each member spent in technical reviews and inspections; how many defects were found. If this information is compared to how much time was spent in testing and how many defects were found, the efficiency of reviews and inspections can be assessed.

(a) *How much time did each member of the team spend on reviews and inspections and how many defects were found?*

This report helps assess the time spent by each member versus the defects found. If too much or too little time is spent, the process has to be re-assessed.

```
SELECT measure."WP_name", measure."Prep_time",
measure."Exam_time", problem."Severity",review."Reviewer_name"
FROM "measure.db" measure , "problem.db" problem ,
"review.db" review
WHERE ( measure.WP_name = problem.WP_name )
AND   ( measure.WP_name = review.WP_name )
GROUP BY  measure."WP_name" , measure."Prep_time" ,
```

```
measure."Exam_time" , problem."Severity" , review."Reviewer_name"
ORDER BY WP_name
```

(b) *How much time was spent in reviews in each phase?*

This report provides information about how much time was spent in reviews in each phase and the number of defects found. This information will help in planning for future projects.

```
SELECT measure."WP_name" , measure."Exam_time" ,
measure."Prep_time" , review."Type" , problem."Severity"
FROM "review.db" review , "measure.db" measure,
"problem.db" problem
WHERE ( measure.WP_name = review.WP_name )
AND ( measure.WP_name = problem.WP_name )
GROUP BY   measure."WP_name", measure."Exam_time" ,
measure."Prep_time",
review."Type" , problem."Severity"
ORDER BY WP_name
```

(c) *How much time was spent in inspections in each phase?*

This report provides information about the amount of time spent in inspections in each phase.

```
SELECT measure."WP_name" , measure."Prep_time" ,
measure."Exam_time", problem."Severity" ,inspection."Type_inspec"
FROM "inspection.db" inspection , "measure.db" measure ,
"problem.db" problem
WHERE ( measure.WP_name = inspection.WP_name )
AND   ( measure.WP_name = problem.WP_name )
GROUP BY   measure."WP_name", measure."Prep_time",
measure."Exam_time",problem."Severity",inspection."Type_inspec"
ORDER BY Type_inspec
```

(d) *How much time was spent in technical reviews and inspections for each work-product?*

This report provides information about the time spent in each phase on each work-product. Too much time spent on a work-product could mean that the work-product is not well understood, or is of low quality. Too little time spent on a work-product could mean that it needs to be further reviewed.

```
SELECT * FROM "measure.db" ORDER BY WP_name
```

(e) *How much time was spent in technical reviews and inspection and how many defects were found?*

This report shows how much time was spent on both inspections and technical reviews in each phase.

```
SELECT measure."WP_name", measure."Prep_time", measure."Exam_time",
measure."Act_rework", measure."Est_rework", measure."No_minorpro" ,
measure."Kind_of_review" ,problem."Severity"
FROM "measure.db" measure , "problem.db" problem
WHERE ( measure.WP_name = problem.WP_name )
GROUP BY   problem."Severity" , measure."Kind_of_review" ,
measure."No_minorpro", measure."Est_rework", measure."Act_rework",
measure."Exam_time", measure."Prep_time", measure."WP_name"
ORDER BY Kind_of_review
```

6.3 Implementation of the quality system

The client uses the web browser (see Figure 6.2) to access javascript forms or reports created using Intrabuilder. When the URL (Universal Resource Locator) is used to request a form or report, the following steps take place:

1. The request is passed to the Web Server which passes it to the IntraBuilder Server.

2. The Intrabuilder server retrieves the required data from the database server, determines the correct formatting, and dynamically creates an HTML page.

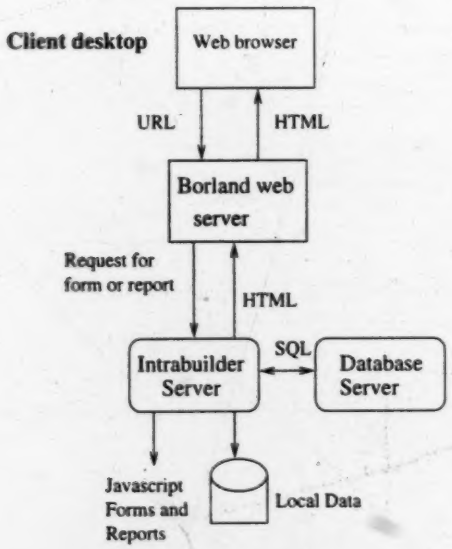

Figure 6.2: Overview

3. The IntraBuilder Server passes the dynamically created HTML page to the Web server.

4. The Web Server passes the HTML page to the Web browser on the client machine which displays it.

6.4 Conclusions

1. Review mechanism

A sequence of forms (see Figure 6.3) support the performance of reviews and inspections using Intrabuilder forms and databases. Performing reviews and inspections will help NewCo verify their work products. This will not only improve the quality of the work products but will also help related processes move to Level 2 (Managed) on the SPICE scale.

2. Metrics and information collection mechanism

Since metrics and other relevant information are collected using the forms, and analyzed using the reports that are generated, related processes will be at Level 4 (Predictable) on the SPICE scale in the area of metrics.

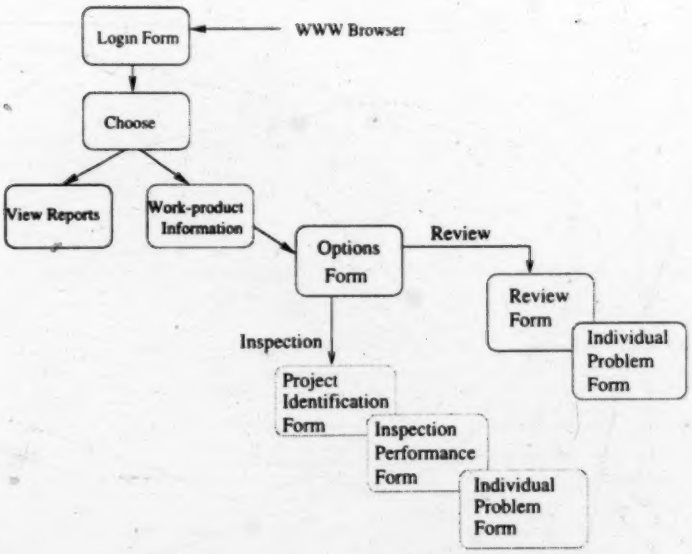

Figure 6.3: Reviews and Inspections

3. Problem tracking mechanism

During technical reviews and inspections, the medium and major problems are tracked individually. Reports will be generated regarding problems that have been discovered during technical reviews/inspections but not solved. Reasons for the causes of the problems are also recorded which will help in process improvement. This problem tracking mechanism will help related processes move up to Level 2 (Managed) on the SPICE scale.

4. Forward and backward traceability

Since the related documents are collected during both technical reviews and inspections, forward traceability will established. Since problems discovered in reviews/inspections of all phases are traced back to previous phases, backward traceability is established too. This will help related processes move up to Level 2 (Managed) on the SPICE scale.

5. Reports Generation

A number of reports which summarize the metrics and information collected during the course of the review will be generated. These reports, described in Section 6.2.2 will help in process improvement. Collecting metrics and analyzing

them for process improvement will help related processes move up to SPICE Level 4 (Predictable).

6. Conforming with the Quality Framework

The quality system implements the components of the quality framework as explained previously. The quality system is :

- *Effective:* The information collected via the forms in the quality system is the information that is generally collected for reviews and inspection in the industry today. The effectiveness of the quality system can only be measured after NewCo uses it and then performs the assessment.
- *Scalable* It is web-based and can be accessed from anywhere via the internet. More databases and forms can be easily created using Intrabuilder[2]. The databases can be stored in a central location and accessed from anywhere in the world.
- *Affordable* The startup costs are low. This is because forms are provided to enter review and inspection information. When these forms are filled in, metrics are collected and reports are generated for human interpretation of quality information. The quality system is deployed over the internet and since most developers today are familiar with WWW browsers, the learning curve is also very low.
- *Usable:* Initial reviews show that the forms that are used to collect information in the quality system are easy to read and understand. Options are provided wherever possible so that the user can just pick one option.
- *Out-sourcable* because the quality system is deployed over the internet and hence can used by any small organization which has access to the internet and has a web browser. It can be easily modified to integrate with another development process.
- *Semi-automated* Users have to manually enter data, but reports are generated automatically and the forms provide an easy means of collecting technical review, inspection and work-product information.

[2]Intrabuilder is a trademark of Borland

Chapter 7

Contributions and Future Research

Most small software organizations do not have the time, money or people to perform extensive quality activities. This thesis has described an approach for introducing process improvement in a small software organization. The improvement strategy is based on a spiral model supported by an ongoing SPICE assessment activity. This thesis focuses on developing a quality framework for small software organizations with the objectives of:

1. defining the characteristics of a small company;
2. assessing a small company to evaluate its current process quality based on a suitable software standard;
3. identifying the areas that need improvement;
4. proposing a quality framework which provides support for quality activities in a small software organization;
5. prototyping a quality system that meets the quality requirements of a specific small company, NewCo, as an instantiation of the quality framework.

7.1 Contributions

Our research successfully fulfilled the thesis objectives with the following contributions:

Quality Framework

A quality framework that is effective, affordable, scalable and usable is proposed in order to provide support for the quality activities in a small software organization.

In order to be effective, the quality system has to evolve incrementally with the development processes of the small organization. It is therefore important to select and use the right development model in developing the product. A survey of existing development models is performed which identifies the advantages and disadvantages of each; and a development model is suggested for use in a small software organization.

Assessment of NewCo

To assist in the refinement of the quality framework, an assessment of NewCo was performed in order to identify problems faced by small software organizations. To this end, a survey of existing standards and their suitability for small organization was performed. This survey identified the problems most existing standards have when applied to small organizations. Of the current available standards, SPICE was deemed the most appropriate for small company assessment.

The SPICE assessment of NewCo identified areas of immediate improvement: traceability, reviews and metrics. A survey of different types of reviews and their use in the different stages of the project was performed.

Prototyped the quality system:

The suggested improvements were prototyped in a tool that provides support for performing the quality activities. The tool is semi-automated and helps in collecting review, work-product and inspection information. Reports are generated for analysis. The prototype provides support for:

1. collecting measurements;
2. performing quality activities such as reviews/inspections;
3. performing individual problem tracking and traceability in support of root cause analysis;

4. providing reports for analysis.

In order to prototype the tool, a survey of the existing technologies was performed that identified the requirements of the tool, and technology that best satisfied the requirements.

7.2 Future Research

Several promising and challenging areas of future research and development related to the development of a quality model for small organizations include:

- *Quality framework model refinement*
  In this thesis, a quality framework model has been proposed that identifies the interfaces between quality and development activities. This model must be refined based on the experiences with NewCo. The interfaces between quality and development activities may also need evolution. With these capabilities, it will be possible to provide services that will enable small companies to *outsource* their quality assessment and improvement activities, leaving them free to focus more on product development.

- *Reviews*
  It is a well recognised fact that reviews help in early detection of defects. It is also well know that defects are more expensive to fix in the later stages of the project. A review model that could suggest the most appropriate kind of review to be performed on the work-product at a given stage in the work-product's evolution would be invaluable. For example, if the work-product is a high risk module, perhaps inspections should be performed, whereas if the work-product is low risk a technical review can be performed. An analysis of which kind of review has to be performed would fit well with the quality model.

- *Quality System Evolution*
  The quality system should evolve so that it incorporates all the potential quality activities performed by small or medium organizations. Parts of the quality system may be activated or de-activated based on the needs of a particular

organization. For example, if an organization is introducing quality activities, and wants to measure only the time taken for each task, the quality system will display forms pertaining to this measurement for the rest of the spiral. In the next spiral, if the organization wants to perform technical reviews or inspections, forms pertaining to these are displayed.

Bibliography

[ANS89] ANSI/IEEE. Ieee standard for software reviews and audits std 1028-1988, June 30 1989.

[BB91] F.W Blackey and M.E Boles. A case study of code inspections. In *Hewlett-packard Journal*, pages 58–63, October 1991.

[BJ94] Judith G. Brodman and Donna L. Johnson. What small businesses say about cmm. In *Proceedings of International conference in Software Engineering(ICSE 16)*, pages 331–340, 1994.

[BL89] David B. Bisant and James.R Lyle. A two-person inspection method to improve programming productivity. In *IEEE Trans. Software Eng., Vol. 15 No. 10*, pages 1294–1304, October 1989.

[BM91] Terry B Bollinger and Clement McGowan. A critical look at Software Capability Evaluations. In *IEEE Software* , pages 25–41, July 1991.

[Boe88a] B Boehm. A spiral model for software development and enhancement. In *IEEE Computer, Vol 21*, pages 61–72, May 1988.

[Boe88b] Boehm, B. A Spiral Model for software development and enhancement. In *IEEE Computer, Vol 21*, pages 61–72, May 1988.

[Boo95] Grady Booch. *Object Solutions - Managing the Object Oriented project.* Addison Wesley Publications, 1995.

[BPV93] M.G. Bradac, D.E. Perry, and L.G. Votta. Prototyping a process experiment. In *Fifteenth International conference on Software Engineering*, May 1993.

[Cd97] Dennis Champeaux de. *Object-Oriented Development Process and Metrics.* Prentice Hall, Upper Saddle River, NJ 07458, 1997.

[CFL95] Fabiano Cattaneo, Alfonso Fuggetta, and Luigi Lavazza. An experience in process assessment. In *17th International conference in Software Engineering, Seattle*, pages 115–121, April 1995.

[Coa94] Francois Coallier. How ISO 9001 fits into the software world. In *IEEE Software, Vol 11 No 1* , pages 98–100, January 1994.

[Coo90] Cooper, Robert. G. Stage Gate Systems: A New Tool for managing New Products. In *Business Horizons* , May/June 1990.

[Coo94] Robert. G Cooper. *Winning at New Products, Second Edition.* Addison Wesley Publications, May 1994.

[cou94] Lloyd's Register TickIT Auditor's course. Issue 1.4, lloyd's register, March 1994.

[Dor93] A Dorling. SPICE: Process Improvement and Capability Determination. In *Information and Software Technology, Vol 35*, June/July 1993.

[ESI96] ESI. Spice. Technical Report SPICE-TI-95-41/0.1.C, European Software Institute, 1996.

[Fag76] M.E Fagan. Design and code inspections to reduce errors in program development. In *IBM Systems Journal*, pages 182–211, Vol 15, No 3, 1976.

[Fow86] P.J Fowler. In-process inspections of work-products at at&t. In *AT&T Technical Journal*, pages 102–112, March/April 1986.

[Fre90] G.M Freedman, D.P. amd Weinberg. *Handbook of Walkthroughs, Inspections and Technical Reviews.* Dorset House, 3 edition, 1990.

[GHP86] M.E Graden, P.S Horsley, and T.C Pingel. The effects of software inspections on a major telecommunications project. In *AT&T technical Journal*, pages 32–40, May/June 1986.

[Goo93] Paul Goodman. *Practical Implementation of Software Metrics.* McGraw-Hill Book Company, 1993.

[Gra92] Robert. B Grady. *Practical Software Metrics for project management and process improvement.* Prentics Hall, Englewood Cliffs, NJ 07632, 1992.

[GS94] R.B Grady and T.V Slack. Key lessons in achieveing widespread inspection use. In *IEEE Software*, pages 46–57, July 1994.

[HMK+94] Volkmar Haase, Richard Messnarz, Gunter Koch, Hans J Kugler, and Paul Decrinis. Bootstrap: Fine-tuning Process Assessment. In *IEEE Software, Vol 11, No 4*, pages 25–35, 1994.

[Hum95] Watts. S Humphrey. *A Discipline for Software Engineering.* Addison-Wesley Publishing company, 1995.

[Inc96] Corporate Marketing Services Inc. Corporate directory. Technical report, Direct Marketing and Mailing Services, 10624 - 169 St. Edmonton, T5P 3X6, 1996.

[ISO86] ISO. Iso quality vocabulary, iso8042, 1986.

[Joh94] P. M Johnson. An instrumented approach to improving software quality through formal technical review. In *Proceedings of 16th International conference in Software Engineering*, pages 113–122. IEEE CS press, 1994.

[KM93] J.C Knight and E.A. Myers. An improved inspection technique. In *Comm. of the ACM*, pages 51–61, Vol 36, No 11, November 1993.

[Koc93] G R Koch. Process Assessment: the Bootstrap approach. In *Information and Software Technology, Vol 35, No 6*, pages 387–403, June/July 1993.

[Kru] Klaus Krull. Intrabuilder architecture overview. White paper (http://www.borland.com/intrabuilder/papers/intraarch/).

[KV96] Amr Kamel and Sundari Voruganti. Assessment Plan for NewCo, October 1996.

[KVHS97] Amr Kamel, Sundari Voruganti, James. H. Hoover, and Paul.G. Sorenson. Software process improvement model for a small organization: an experience report. In *Annual Oregon Workshop on Software Metrics 97, Coeur d'Alene, Idaho*, May 1997.

[MDTR93] Vahid Mashayekhi, M.J. Drake, Wei-Tak Tsai, and John Riedl. Distributed, collaborative software inspection. *IEEE Software*, pages 66–75. September 1993.

[Mil88] E.E Mills. Software metrics, sei curriculum module sei-cm-12-1.1. Technical report, Carnegie Mellon University, Pttsburg, PA, 1988.

[MW91] Tim Maude and Graham Willis. *Rapid Prototyping - The management of software risk.* Pitman Publishing, 1991.

[Ole96] Tony Olekshy. Planning considerations. Document AGO/CRK/WOOP Redaction 3.0.6 file:/usr/tees1/misc/seaf/seaf-doc/plancons/section/document.htm, April 1996.

[Pre92] Roger. S. Pressman. *Software Engineering - a practioner's approach.* McGraw-Hill Inc, New York, 3 edition, 1992.

[PW87] D.L. Parnas and D.M. Weiss. Active design reviews: Principles and practices. In *J. of Systems and Software, No 7*, pages 259–265, 1987.

[PWG93] M. C. Paulk, C. V. Weber, and S. M. et al. Garcia. Key Proctices of the Capability Maturity Model Ver 1.1. Technical report, Software Engineering Institute, 1993.

[Roy70] W. W Royce. Managing the development of large software systems: Concepts and Techniques. In *Proceedings WESCO* , 1970.

[Rus91] G.W Russel. Experience with inspection in ultralarge scale developments. In *IEEE Software*, pages 25–31, January 1991.

[SC94] Joc Sanders and Eugene Curran. *Software Quality.* Addison-Wesley, 1994.

[SE93] Susan. H. Strauss and Robert. G. Ebenau. *Software Inspection Process.* McGraw Hill Inc., 1993.

[Shi92] G.C Shirey. How inspections fail. In *Proc. 9th International conf. Testing computer software*, pages 151–159, 1992.

[Sil92] B Silver. TQM vs the SEI Capability Maturity Model. In *Software Quality World, Vol 4, No 2*, December 1992.

[Sir82] Siropolis, N. C. . *Small Business Management* . Houghton Mifflin Company, Boston, 2 edition, 1982.

[SK95] Hossein Saiedian and Richard Kuzaran. SEI Capability Maturity Model's impact on contractors. In *IEEE Computer*, pages 16–26, January 1995.

[SS88] Andrew J Szonyi and Dan Steinhoff. *Small business management fundamentals.* McGraw-Hill Ryerson, 1988.

[Wal94] Ernest Wallmuller. *Software Quality Assurance - A practical approach.* Prentice Hall Publications, 1994.

[WBM96a] D Wheeler, B Brykczynski, and R Jr. Meeson. Peer review processes similar to inspection. In *Software inspection - an industry best practice*, pages 228 – 236. IEEE Computer Society, 1996.

[WBM96b] D Wheeler, B Brykczynski, and R Jr. Meeson. *Software inspection - an industry best practice.* IEEE Computer Society, 1996.

[Web88] Webster. New lexicon webster's encyclopedic dictionary of the english language, 1988.

[Wei91] Gerald. M Weinberg. *Quality software management - First-Order measurement*, volume 2. Dorset House, 1991.

[Wel93] E.F Weller. Lessons from three years of inspection data. In *IEEE Software*, pages 38–45, September 1993.

[You89] Edward Yourdon. *Structured Walk-throughs.* Prentice-Hall, Englewood Cliffs, N.J., 4 edition, 1989.

Appendix A

User Manual

This appendix provides an overview of the important forms of the system. The Quality System can be accessed through the Uniform Resource Locator (URL): *http://peoria.cs.ualberta.ca/seaf/review.htm*. In order to prevent unauthorized access. the users are required to login.

A.1 Login Form

- When the user first lo. s in to the Quality System. a *login form* (see Figure A.1) is displayed.
- The user types in the login name and password. This helps to prevent unauthorized users from crashing into the system.
- An explanation of the fields on the form is given in table A.1.
- Clicking on *Submit Login* will verify the user name and password.
  - If the user name and password match the ones stored in the User Table. the *Options Form* (see Figure A.2) is displayed.
  - If not. the text on the button changes to **Try Again**.

A.2 Options Form

This form gives the users a choice of:

Figure A.1: Login Form

Name	Reason
Login Name	The users login
Password	The users password to validate the access

Table A.1: Login Form

- *Viewing reports:* These reports are pre-generated and categorized.
- *Work-product Information:* This form has information about all the work-products in the database and allows creation of new work-products and editing of existing ones.
- *Change/Query:* This form displays all the problems and information regarding them. The problem status may be changed.

A.3 Work-Product Information

This form (shown in figure A.3) allows the users to select existing work-products. editing of existing work-products or entering new work-products into the database. An explanation of the fields is given in Table A.2.

Name	Reason
Project Name	name of the project the work-product belongs to
Work-product name	name of the work-product
Related work-products	work-products related to current one
Work-product Type	Specifies if the work-product is a document or code
Kind of work-product	Specific kind of work-product
Work-product status	Specify the status of work-product
Size	The size of work-product in pages or lines
Project Name	The project to which the work-product belongs
Author	Author of work-product

Table A.2: Work-product Form

Clicking *OK* here will display the *Choose Option* form which allows input of review and inspection related information.

A.4 Choose the Review

This form gives the user an option to enter review or inspection information. Figure A.4 shows how the form looks.

Clicking on *Input Review Information* displays the *Review Form*. Clicking on *Input Inspection Information* displays the *Project Identification Form* (See figure A.7).

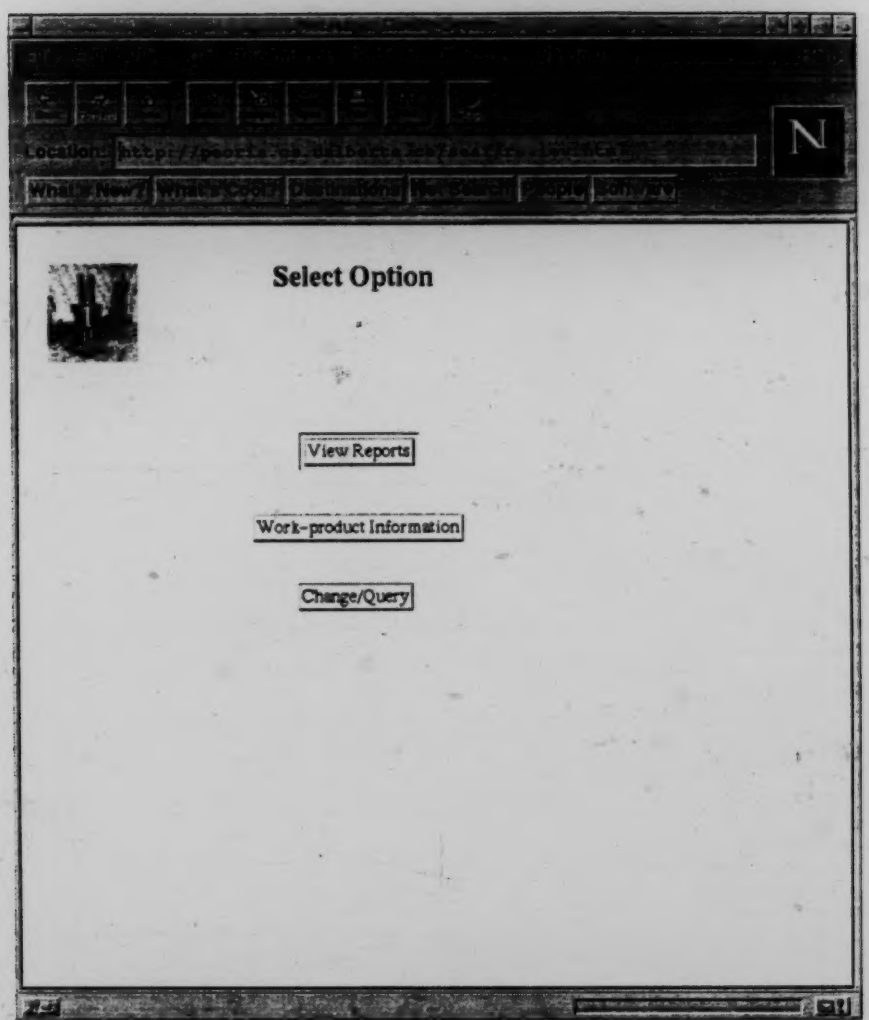

Figure A.2: Options Form

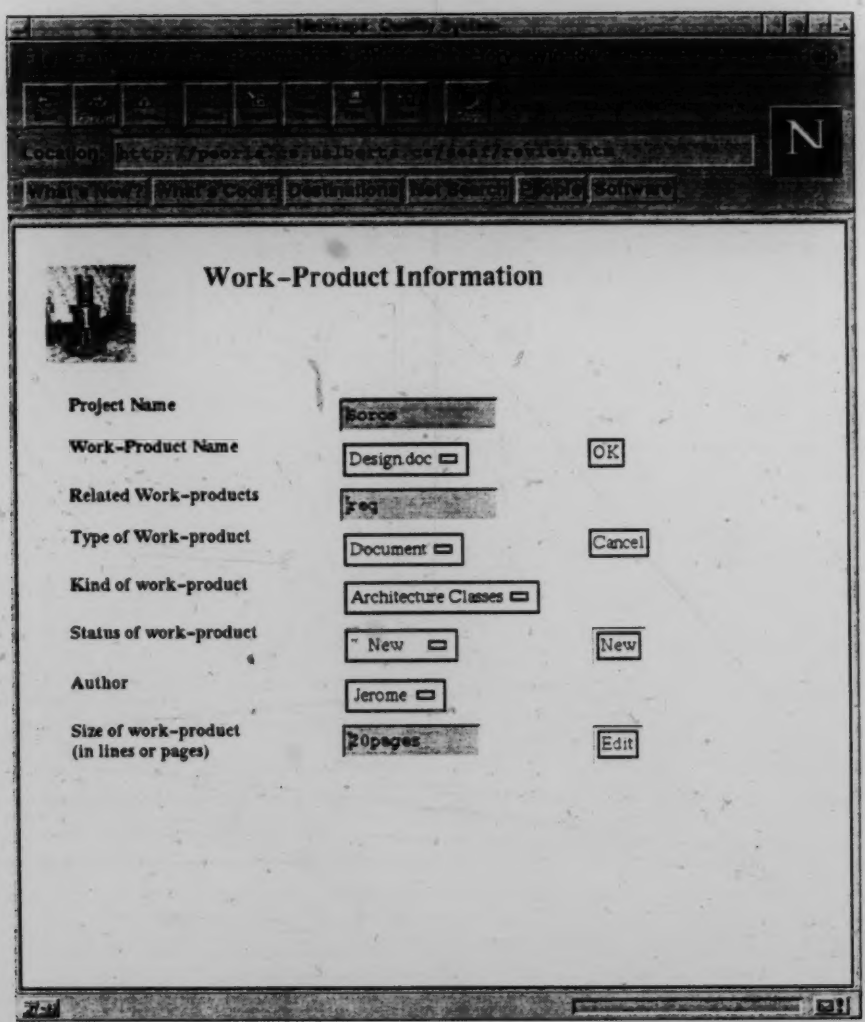

Figure A.3: Work-Product Form

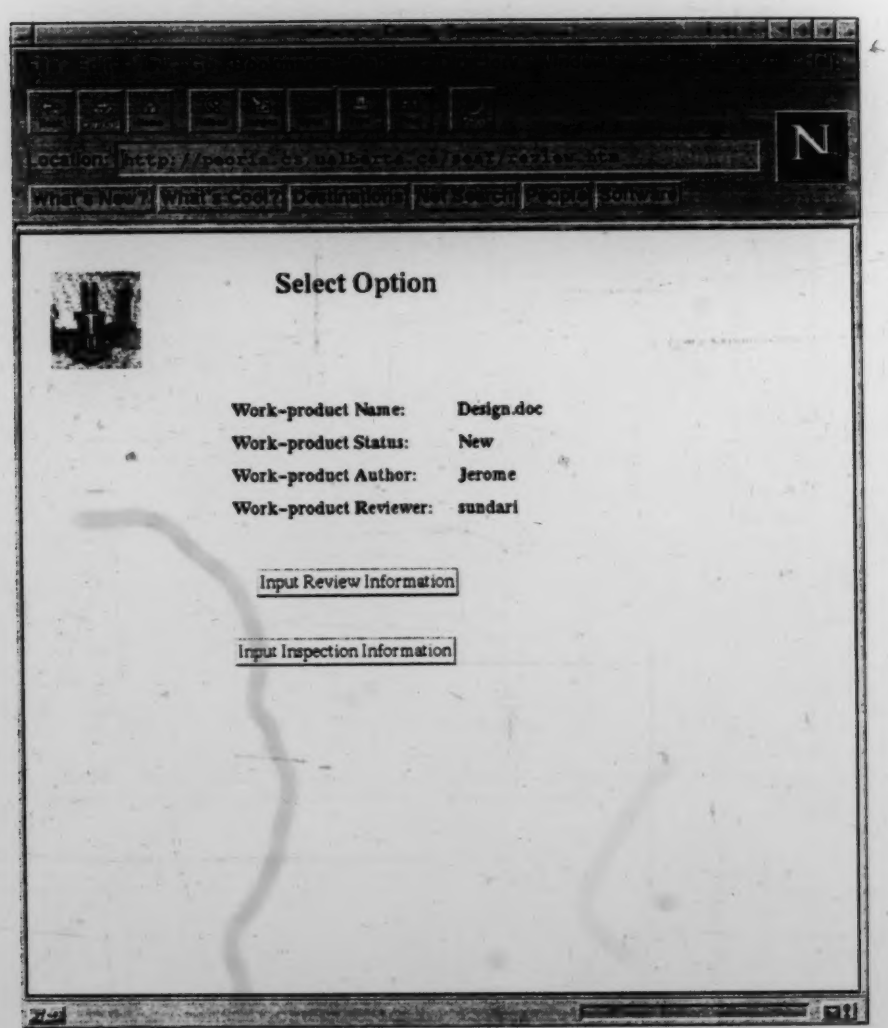

Figure A.4: Choose Form

A.5 Review Process

The *Review Information Form* (shown in figure A.5) collects information about the document under review.

A.5.1 Review Form

- All minor problems (like spelling mistakes) are reported in a document whose file name can be specified in this form. The number of minor problems can also be specified.
- Clicking on *Done* will save the information to the database and the *Individual Problem Form* will be displayed (Figure A.6).
- Clicking on *Cancel* abandons the editing and returns to the previous *Choose Options Form.*

A.5.2 Individual Problem Form

The *individual problem form* (See table A.3 for an explanation) records the serious problems that were discovered during the review. These problems may be categorized as major problems or medium problems based on their severity. This form allows the user to enter the description for the problem and the reasons for it.

- But the medium and major problems are recorded individually using the *Individual Problem Form* (see Figure A.6) and tracked.
- Clicking on *Done* will save the problem information to the database and exit the form.
- Clicking *Next Problem* will save the current problem information to the database and display an empty form to fill in for the next problem.

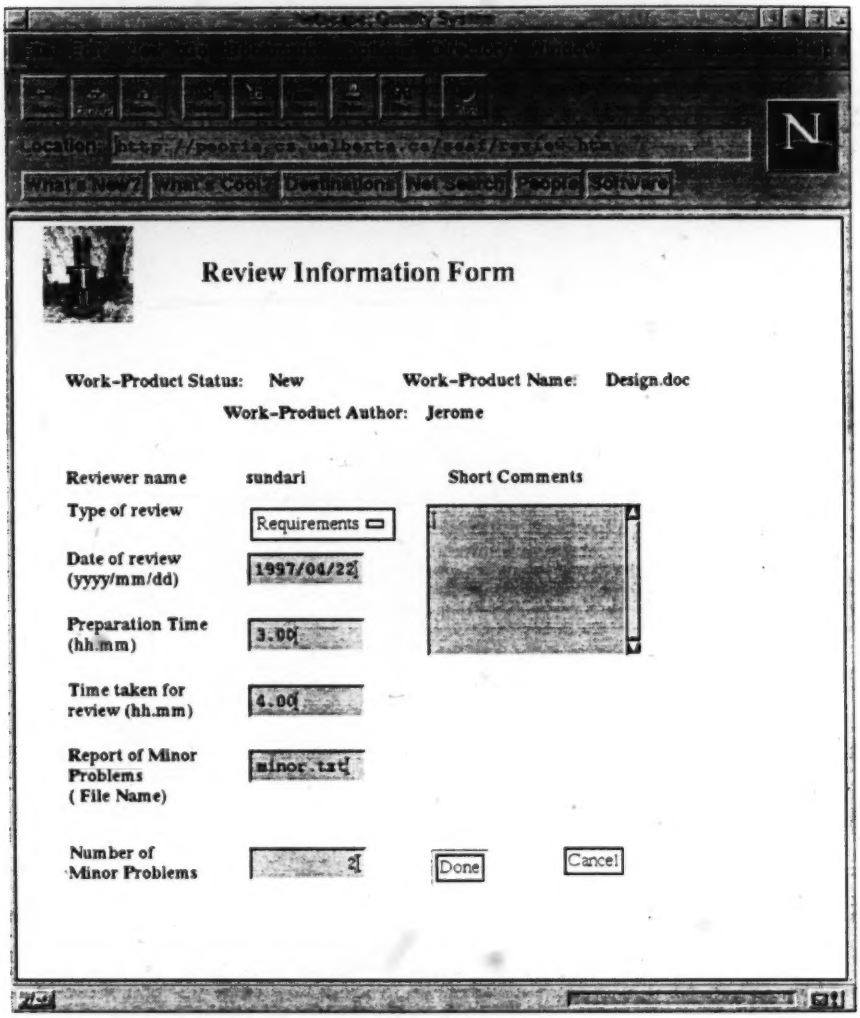

Figure A.5: Review Information Form

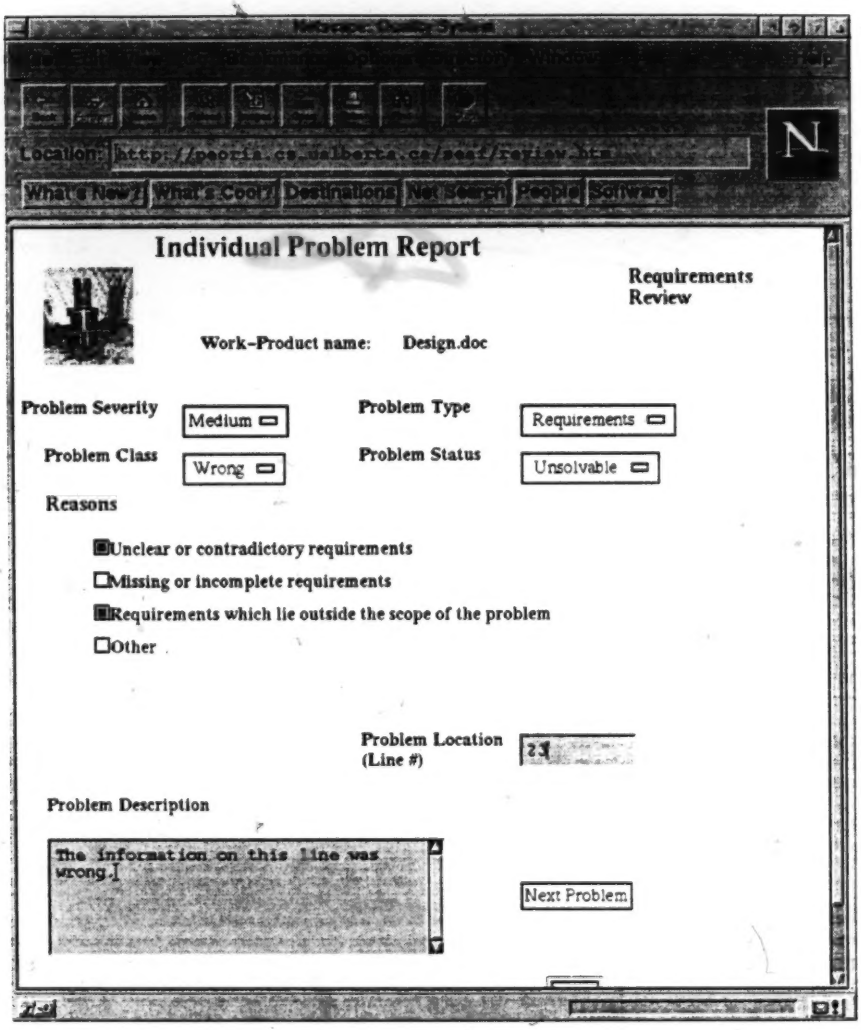

Figure A.6: Individual Problem Form

Name	Reason
Problem Severity	If the problem is major or minor
Problem Type	The source of the problem: reviews or customer reports
Problem Class	Missing means more information needed. wrong means wrong information used and extra means more information is needed
Problem status	is the status of the problem: open, closed or next release
Reasons	The reasons for the problem
Problem Location	the line number the problem occured in
Problem description	The detailed description of the problem

Table A.3: Problem Form

A.6 Inspections

A.6.1 Project Identification Form

This form collects project specific information. All the Inspection Forms collect information that is specified by Fagan.

- Clicking on *Post* will save the information to the database and display the *Inspection Performance Form* (see Figure A.8).
- Clicking on *Cancel* will abandon the edit mode and return to the *Options Form.*

A.6.2 Inspection Performance Form

The Inspection Performance Form (as shown in FigureA.8) collects data about how long the inspections took.

- Clicking on *Done* will save the data to the database and display the *Individual Problem Report Form* (see Figure A.9) to enter defect related information.
- Clicking on *Cancel* will abandon the edit and go back to the *Project Identification Form.*

A.6.3 Individual Problem Report Form

The Individual Problem Report Form (shown in Figure A.9) collects information about each defect. Most inspections classify defects as minor defects or major de-

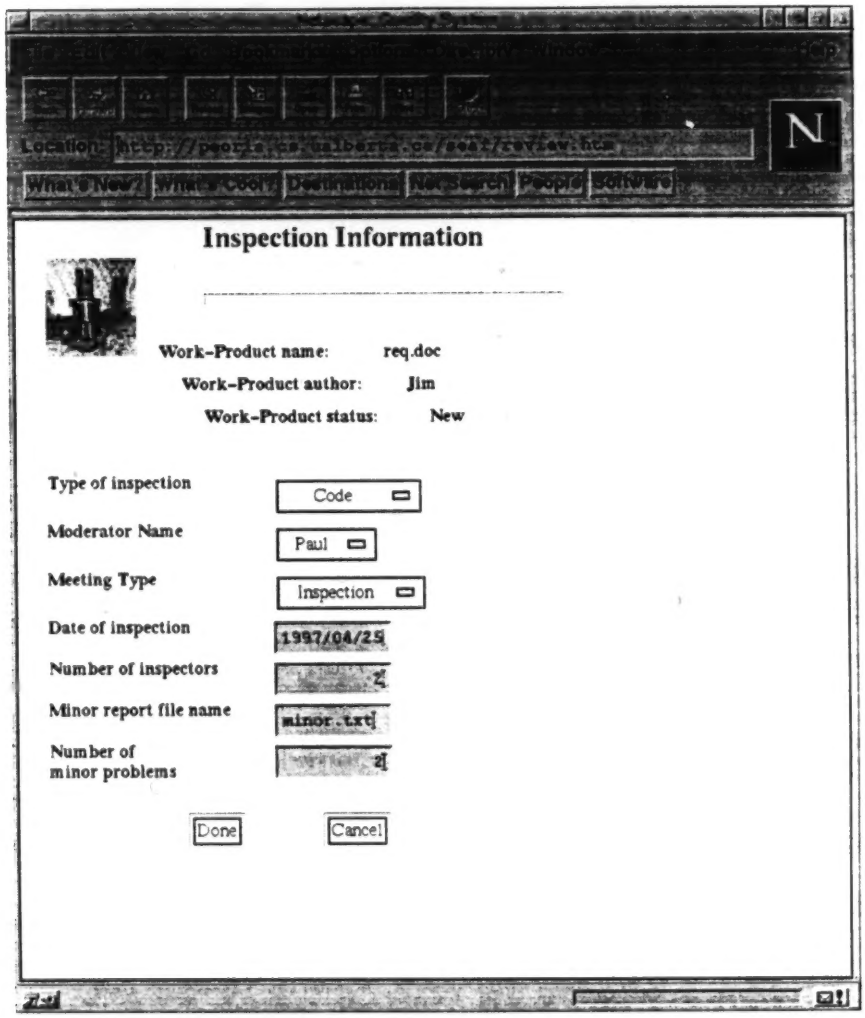

Figure A.7: Project Identification Form

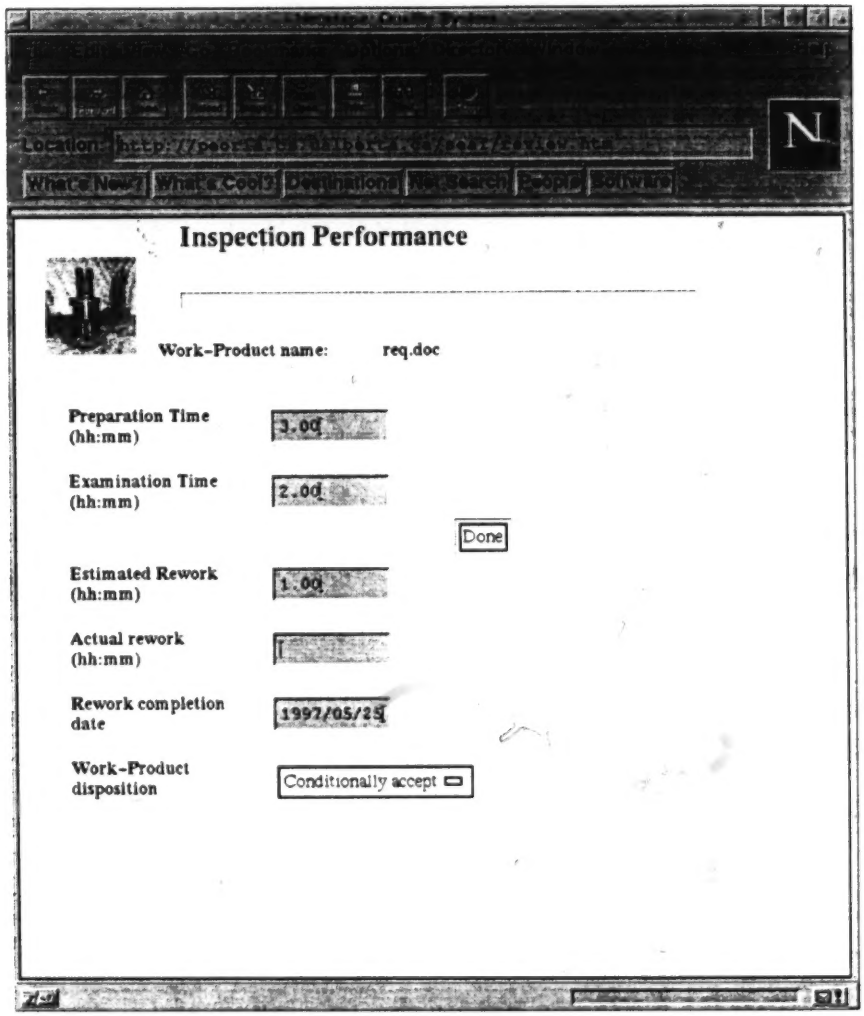

Figure A.8: Inspection Performance Form

fects. Since minor defects are spelling mistakes and trivial mistakes, they could be all specified in a report. But the medium and major defects are non-trivial defects and will be tracked individually.

- Clicking on *Done* will save the data to the database.
- Clicking on *Next* will save the current defect description and move on to the next one.

A.7 Reports

Reports are invoked when the *Reports* button is clicked on the *Options* form. The reports form (as shown in Figure A.10) displays the different categories of reports available. For each category, a new form is displayed and the possible reports can be checked prior to display. See the *Process Improvement Reports* from Figure A.11 as an example.

A.8 Change/Query Form

When the *Change/Query* button on the *Options* forms is clicked, the change/query form is displayed. This form displays all information related to problems in the database. It allows changing of status of a problem. Problems can also be queried on (See Figure A.12).

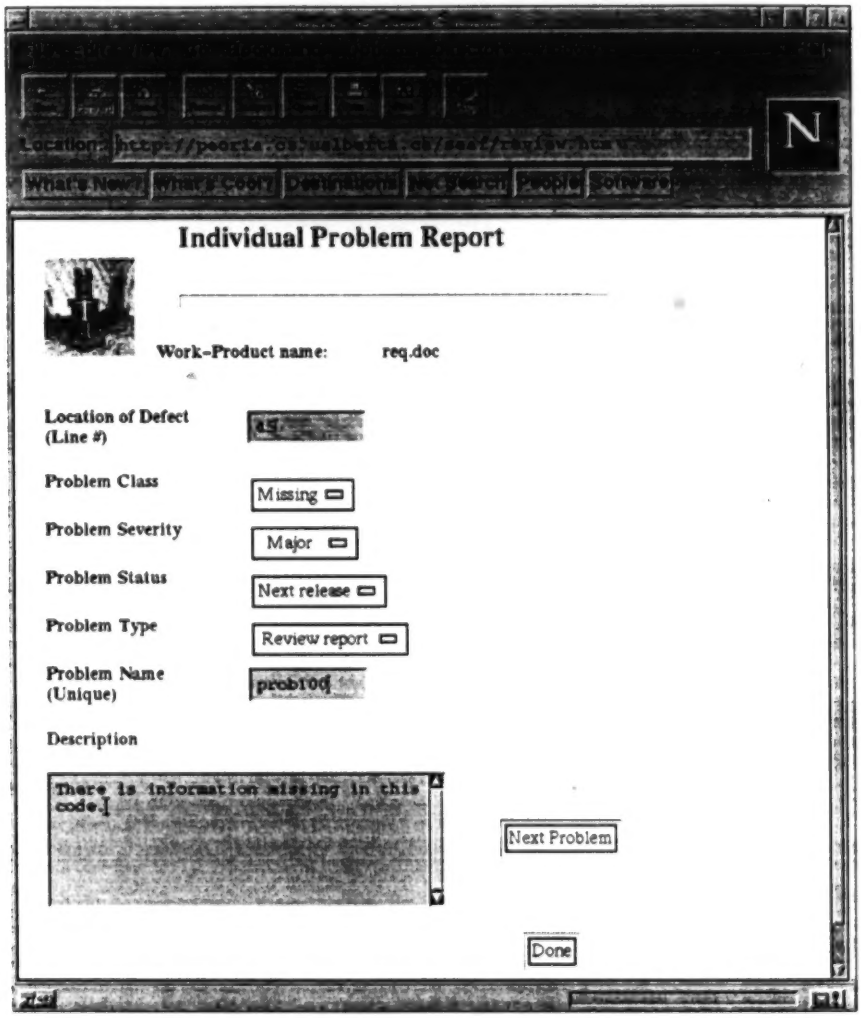

Figure A.9: Defect Form

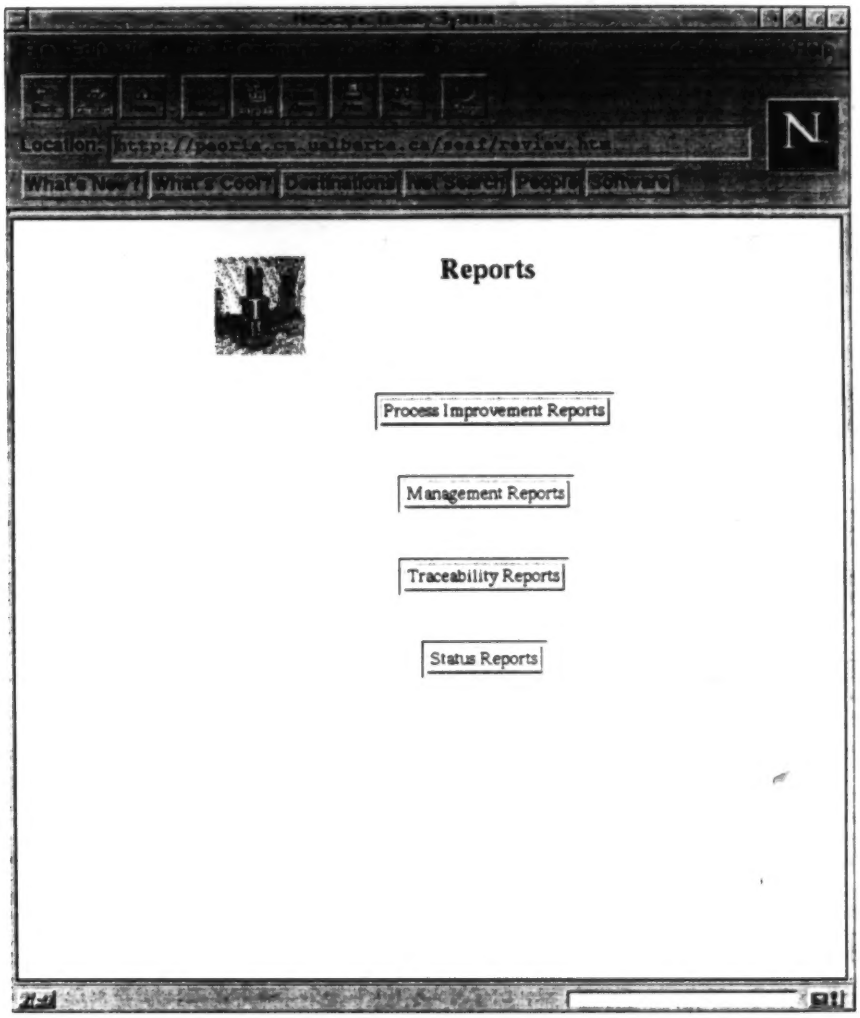

Figure A.10: Reports

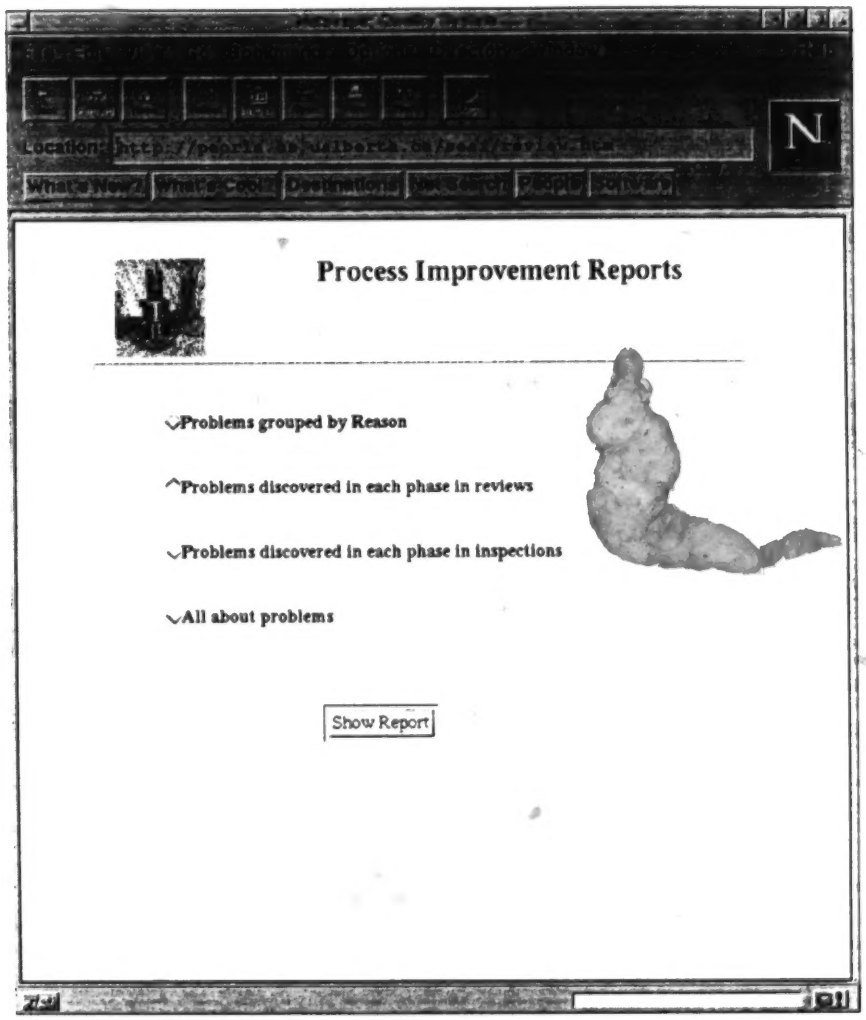
Process Improvement Reports
Problems grouped by Reason
Problems discovered in each phase in reviews
Problems discovered in each phase in inspections
All about problems
Show Report

Figure A.11: Process Improvement Reports

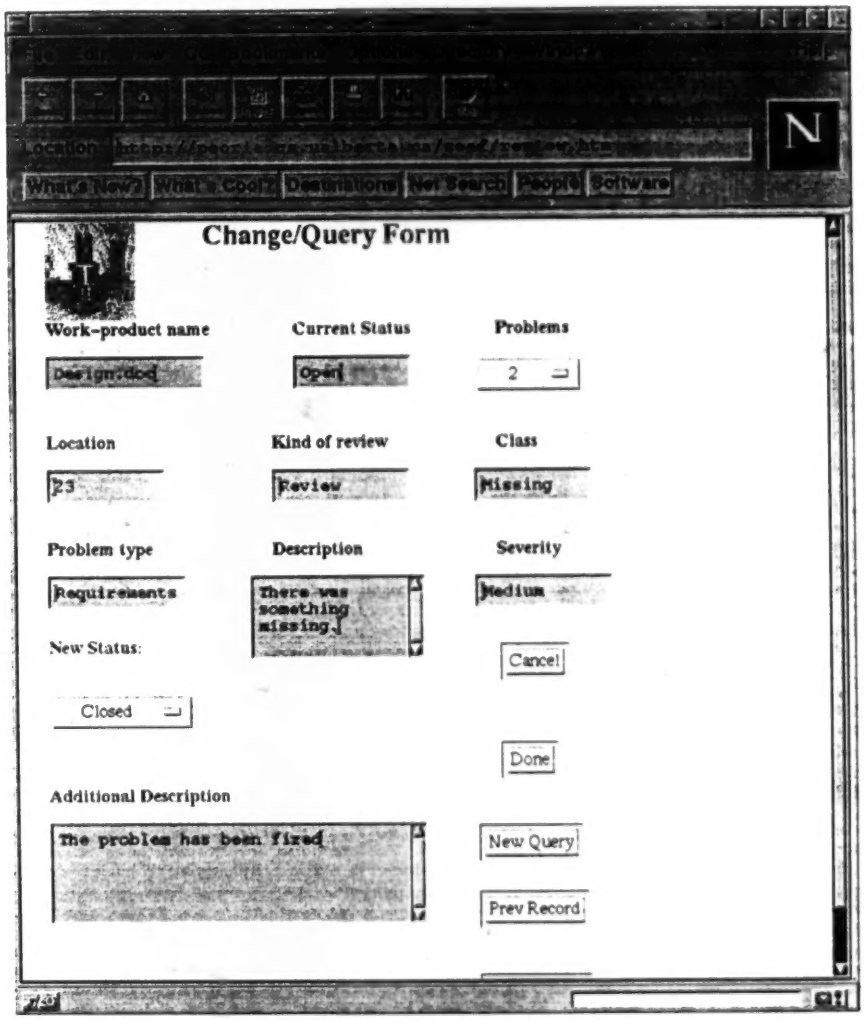

Figure A.12: Change/Query Form

Appendix B

Technology Review

This appendix provides an overview of the tools under review for prototyping the quality system.

B.1 Requirements for the quality system

- A tool that can be used by a group of people. E-mails must be sent automatically when events such as changes in documents occur.
- A database back-end to store metrics and traceability information.
- A system that the developers do not have to spend appreciable effort to learn. That is, a system that is fairly easy to use and integrated with the developer's environment.
- A tool that allows posting of comments relevant to a document/code.
- A tool that allows forms and templates to be easily created.
- A tool that allows data to be entered into the database in addition to displaying information in the database.
- A tool that provides newsgroups or newsgroup-like facilities.

B.2 Tools under review

B.2.1 Lotus Notes

Lotus Notes is a groupware package that is currently being used to release versions, discuss issues (design etc.) and as a medium of communication. In order to develop a quality framework in Lotus Notes, the following are the perceived advantages:

- It has a database where documents can be posted and changes can be just responses to the main document.
- Databases for different discussion issues are easy to create and use.
- Work-flow implementation is fairly easy.
- Notes can be integrated into the developer's environment using agents that watch for a specific event to happen.
- Replication, mail and security are automatically taken care by Lotus Notes.
- The setup is fairly simple - only one server and a number of clients.

The following are the perceived disadvantages:

- Documents more than a few pages long are not easy to write.
- Notes database does not facilitate collection of metrics and traceability information. This has to be done in a database engine and then linked to Notes (if possible). Database classes can be used to link Notes with external databases, but it is a one-way connection. The databases cannot be updated using Notes forms and can only be displayed.

B.2.2 Intranet

An intranet is a company specific internet. There are a number of software packages that help in building the intranet. The current one being assessed is *Intrabuilder* from Borland.

The perceived advantages of using an intranet are:

- The quality system is more likely to be used because developers are more familiar with web browsers. Code check-in and check-out can be performed using web browsers, or a configuration management system can be integrated.
- Intrabuilder has Borland's database engine as a back-end which supports all the database needs.
- Intrabuilder provides a two-way link between the database and the HTML documents displayed on the web. That is, data can be displayed from the database and data can be entered into the database.
- There are a number of wizards that make building the application very easy and fast.

The perceived disadvantages are:

- It is very easy to categorize documents in Lotus Notes. If an intranet is constructed, newsgroups have to be created for discussions on each document. This might make handling information more difficult.
- Replication-like functionality is not provided and has to be coded.
- A news server has to be used to manage the news groups. Security issues for the newsgroups have to be considered.
- A number of servers have to be used -
  - News server
  - Web server
  - Mail server
  - Intrabuilder server
- E-mail has to be handled separately.

Another Intranet solution that is being considered is the Netscape Communicator and Netscape SuiteSpot.

Advantages:

- The Communicator products are very useful. See
  http://home.netscape.com/comprod/products/communicator/index.html
  for more details.

Disadvantages:

- The Livewire product uses an Informix backend, which is very expensive.
- A number of servers have to be used.